
SONDERHEFT 69 Markt&Technik öS 120,-/sfr. 16,-/Lit. 16000 hfl. 21,-/dkr. 75,-/fmk 62,- DM 16,-
DISKETTE IM HEFT
64'er
BASIC
Zeichnen, Malen, Konstruieren
110 neue Befehle für Grafik und Sprites
Workshop
Großer Basic-Kurs zum Mitmachen
Tips & Tricks
So programmieren Profis
Super-Tool
XBasic+: Kraftpaket für Programmierer
58 Programme auf Diskette
64'er
PRINT
NEXT
IF
FOR
GO TO

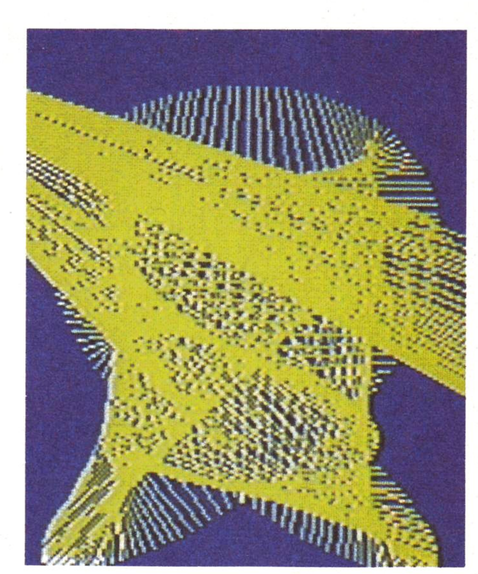


Basic-Kurs

Knobelecke

Tips & Tricks

Basic-Erweiterungen



Grafik: Basic-Erweiterung der Extraklasse: Sprite+Grafik-Basic Seite 38

Sonstiges

Alle Programme aus Artikeln mit einem [Disketten]-Symbol finden Sie auf der beiliegenden Diskette (Seite 19)

**** COMMODORE 6
64K RAM SYSTEM 389
READY.

Basic-Kurs zum Mitmachen

BASIC 2.0 nicht NUR FÜR EINSTEIGER

von Dr. H. Hauck/
Ch. Bergmann/H. Beiler

Keine Angst vorm Programmieren! Es muß ja nicht gleich Maschinensprache sein... Staunen Sie, was man mit Basic alles anstellen kann.

Basic, Abkürzung von »Beginners All-purpose Symbolic Instruction Code«, bedeutet wörtlich übersetzt: Einsteiger-Programmiersprache für jeden Zweck.

Auf den Punkt gebracht: Auch Basic läßt sich so geschickt einsetzen, daß es vielen Maschinenprogrammen kaum nachsteht – abgesehen von der Geschwindigkeit. Die Gründe sind bekannt: Die meiste Zeit für ein Basic-Programm vergeudet der Computer mit der Übersetzung der Befehle (z.B. PRINT, GOTO, GOSUB usw.) durch den Basic-Interpreter. Das ist der Preis, den man für diese klar verständliche Programmiersprache bezahlen muß: Jeder Basic-Befehl wird nämlich zuerst vom Computer identifiziert, eingeordnet und die entsprechende Maschinensprache-routine im Betriebssystem aktiviert – ein Haufen Action! Trotzdem ist Basic für die meisten C-64-Besitzer die einzige Kommunikationsmöglichkeit mit dem Computer. Wenn man außerdem ein Basic-Programm gut strukturiert, umständliche Routinen vermeidet und sauber programmiert, ist bei normalen Anwendungsprogrammen (z.B. Dateiverwaltung, Text- und Bildschirmausgaben usw.) gar nicht so leicht einzusehen, warum der Basic-Programmierer unbedingt auf Assembler umsteigen sollte.

Bei unserem Kurs setzen wir voraus, daß Sie die Schwellenangst überwunden und sich doch schon mal als Programmautor versucht haben. Ebenso sollten Sie sich in den Tastaturfunktionen und der Speicherbelegung des C64 einigermaßen auskennen. Wir wollen aber nicht die Informationen des Handbuchs zum C64 neu aufwärmen, sondern sie ergänzen!

Zu unserem Kurs finden Sie eine Menge Beispielprogramme auf der Diskette zum Sonderheft. In übersichtlichen Kurzinfos erläutern wir, was das Programm macht. Die Bezeichnungen der Kurzinfos stimmen exakt mit den Dateinamen auf der Diskette überein.

4 BASIC V2 ****

11 BASIC BYTES FREE

Interpretationen

Basic ist eine Dialogsprache. Mit der Tastatur kann man den Computer direkt ansprechen: Seine Antwort erscheint auf dem Bildschirm. Dazu existiert eine Reihe von Befehlen, die man als **direkte** Kommandos eingeben kann. Sie werden unmittelbar nach Tipp auf die RETURN-Taste ausgeführt: PRINT, LOAD, SAVE, OPEN, CLOSE, VERIFY, RUN, LIST, NEW usw.). Im **Programmodus** arbeitet der Computer Anweisungen ab, die im dafür vorgesehenen Speicher (Basic-RAM) stehen. Befehle und Anweisungen sind in Programmzeilen untergebracht, die mit einer Zeilennummer beginnen. Nach dem Start mit RUN sortiert der Computer die Zeilen in aufsteigender Reihenfolge und führt die Befehle Schritt für Schritt aus.

Um dies zu steuern, besitzt der C64 ein umfangreiches Programm in Maschinensprache (etwas anderes versteht er nicht!), den **Basic-Interpreter**. Er liegt im ROM-Bereich von Adresse 40960 bis Speicherstelle 49151. Dieses 8 KByte große Assembler-Programm untersucht Ihre Basic-Eingaben und überprüft sie anhand einer Tabelle im Bereich von 41118 bis 41372 (**Abb. 1**) auf die richtige Schreibweise (Syntax). Vor allem Anfänger kennen den Effekt: Wenn etwas nicht stimmt, erscheinen die berühmten Fehlermeldungen (**Abb. 2**). Die entsprechenden Texte befinden sich unmittelbar hinter den Befehlswörtern: von Adresse 41374 bis 41757. Gab's nichts auszusetzen, wandelt der Interpreter die Befehle in ihre Kurzform um: Tokens. Sie umfassen lediglich 1 Byte mit Werten zwischen 128 und 203. In einer Tabelle (von 40972 bis 41087) ist erfaßt, ab welchen Adressen die verschiedenen Maschinenroutinen liegen, die für die Ausführung der Basic-Anweisung verantwortlich sind (**Tabelle 1**). Nach getaner Arbeit kehrt der Basic-Interpreter wieder in die Hauptschleife zurück und wartet auf neue Arbeit.

Zwei Basic-Programme auf der Diskette zu diesem Sonderheft lesen die Text-Bytes des Basic-Interpreters und geben sie auf dem Bildschirm aus: **basic-befehle** und **fehlermeldungen**. Man lädt sie wie jedes Basic-Programm und startet sie mit RUN. Der Bildschirm bringt die Befehlswörter bzw. die Basic-Fehlermeldungen formatiert. Eines fällt auf: Der letzte Buchstabe ist stets groß (Versal). Damit weiß der Basic-Interpreter, daß das Befehlswort zu Ende ist.

Der letzte, zusammenhängende Textbereich des Basic-Interpreters ist die Einschaltmeldung. Sie be-

```
enD      pokE     ↑      vaL
foR      print#   anD    asC
nexT     prinT    oR     chr$
datA     conT     <      left$
input#   lisT     =      right$
inpuT    clR      >      mid$
diM      cmD      sgN    gO
reaD     syS      inT
leT      opeN     abS
gotO     closE    usR
ruN      geT      frE
iF       neW      poS
restorE  tab(     sqR
gosuB    tO       rnD
returN   fN       loG
reM      spc(     exP
stoP     theN     coS
oN       noT      siN
waiT     steP     taN
loaD     +        atN
savE     -        peeK
verifY   *        leN
deF      /        str$
```

[1] Diese Befehle kennt der Basic-Interpreter des C64

```
too many fileS
file opeN
file not opeN
file not founD
device not presenT
not input filE
not output filE
missing file namE
illegal device numbeR
next without foR
syntaX
return without gosuB
out of datA
illegal quantitY
overfloW
out of memorY
undef'd statemenT
bad subscripT
redim'd arraY
division by zerO
illegal direcT
type mismatcH
string too lonG
file datA
formula too compleX
can't continuE
undef'd functioN
```

[2] Hinweise für Programmierer: die Fehlermeldungen

Tabelle 1. Tokens und Betriebssystemadressen der Basic-Befehle

Anweisung/Befehl	Adresse hex./dez.	Token hex./dez.
END	A831 (43057)	80 (128)
FOR	A742 (42818)	81 (129)
NEXT	AD1E (44318)	82 (130)
DATA	A8F8 (43256)	83 (131)
INPUT#	ABA5 (43941)	84 (132)
INPUT	ABBF (43967)	85 (133)
DIM	B081 (45185)	86 (134)
READ	AC06 (44038)	87 (135)
LET	A9A5 (43429)	88 (136)
GOTO	A8A0 (43168)	89 (137)
RUN	A871 (43121)	8A (138)
IF	A928 (43304)	8B (139)
RESTORE	A81D (43037)	8C (140)
GOSUB	A883 (43139)	8D (141)
RETURN	A8D2 (43218)	8E (142)
REM	A93B (43323)	8F (143)
STOP	A82F (43055)	90 (144)
ON	A94B (43339)	91 (145)
WAIT	B82D (47149)	92 (146)
LOAD	E168 (57704)	93 (147)
SAVE	E156 (57686)	94 (148)
VERIFY	E165 (57701)	95 (149)
DEF	B3B3 (46003)	96 (150)
POKE	B824 (47140)	97 (151)
PRINT#	AA80 (43648)	98 (152)
PRINT	AAA0 (43680)	99 (153)
CONT	A857 (43095)	9A (154)
LIST	A69C (42652)	9B (155)
CLR	A65E (42590)	9C (156)
CMD	AA86 (43654)	9D (157)
SYS	E12A (57642)	9E (158)
OPEN	E1BE (57790)	9F (159)
CLOSE	E1C7 (57799)	A0 (160)
GET	AB7B (43899)	A1 (161)
NEW	A642 (42562)	A2 (162)
SGN	BC39 (48185)	B4 (180)
INT	BCCC (48332)	B5 (181)
ABS	BC58 (48216)	B6 (182)
USR	0310 (784)	B7 (183)
FRE	B37D (45949)	B8 (184)
POS	B39E (45982)	B9 (185)
SQR	BF71 (49009)	BA (186)
RND	E097 (57495)	BB (187)
LOG	B9EA (47594)	BC (188)
EXP	BFED (49133)	BD (189)
COS	E264 (57956)	BE (190)
SIN	E26B (57963)	BF (191)
TAN	E2B4 (58036)	C0 (192)
ATN	E30E (58126)	C1 (193)
PEEK	B80D (47117)	C2 (194)
LEN	B77C (46972)	C3 (195)
STR$	B465 (46181)	C4 (196)
VAL	B7AD (47021)	C5 (197)
ASC	B78B (46987)	C6 (198)
CHR$	B6EC (46828)	C7 (199)
LEFT$	B700 (46848)	C8 (200)
RIGHT$	B72C (46892)	C9 (201)
MID$	B737 (46903)	CA (202)
Basic-Operatoren		
OR	AFE6 (45030)	46 (70)
AND	AFE9 (45033)	50 (80)
NOT	AED4 (44756)	5A (70)
=	B016 (45078)	64 (100)
+	B86A (47210)	79 (121)
-	B853 (47187)	79 (121)
*	BA2B (47659)	7B (123)
/	BB12 (47890)	7B (123)
7	BF7B (49019)	7F (127)

Kurzinfo: basic-befehle

Nach dem Start mit RUN gibt der Bildschirm eine in vier Spalten formatierte Liste aller Befehlswörter des Basic 2.0 aus.

Zeile 9: Vorbesetzung der Variablen, Bildschirm löschen und Kleinschrift einschalten (wird durch ein reverses N = CHR$(14) erzeugt). Z ist die Zählvariable für die aktuelle Bildschirmzeile, X und TB kümmern sich um die aktuelle Spalte: Zu Beginn gilt der Wert 2.

Zeile 10: In einer FOR-NEXT-Schleife liest der Computer die Speicherinhalte der ROM-Adressen im Basic-Interpreter von 41118 bis 41372 und speichert den jeweiligen Wert (CHR$) in der Variablen A$.

Zeilen 11 bis 21: Manche Codes, vor allem Rechen- und Vergleichsoperatoren, sind im Interpreter-ROM mit einem um »128« ($80) höheren Wert ausgestattet. Die IF-THEN-Abfragen dieser Basic-Zeilen ändern die Codes für eine korrekte Bildschirmausgabe.

Zeile 28: Diese Abfrage überprüft, ob Z bereits den Wert 23 angenommen hat. Falls ja, setzt das Programm Z wieder auf 0, erhöht aber X um einen Zähler. X wird mit 10 multipliziert und das Rechenergebnis an TB weitergegeben: Das ist die jetzt aktuelle Cursorspalte für die Bildschirmausgabe. Die folgenden POKE- und SYS-Anweisungen geben die Werte an den Computer weiter: Adresse 214 = Zeile, 211 = Spalte.

Zeile 29: Hier macht sich das Programm zunutze, daß jedes Basic-Befehlswort als Endekennzeichen einen Großbuchstaben besitzt. Solche Zeichen besitzen immer einen Wert, der größer als »127« ist. Trifft diese Bedingung zu, wird an den Großbuchstaben noch ein <RETURN> (CHR$(13)) angefügt, damit das nächste Wort in der nächsten Zeile steht. Dadurch erhöht sich auch der Zeilenzähler Z um »1«.

Zeile 39: Der aktuelle Inhalt von A$ wird an der momentanen Spaltenposition (TB) ausgegeben. Das an die Variable angefügte Semikolon sorgt dafür, das die aus dem ROM-Speicher gelesenen Zeichen hintereinander auf dem Bildschirm erscheinen (und nicht untereinander!).

Zeile 40: NEXT erhöht die Schleifenvariable I um »1« und läßt den Computer zu Zeile 10 zurückkehren. Das Spiel beginnt von neuem, bis die Speicherstelle 41372 erreicht ist.

Zeile 50: Wenn die FOR-NEXT-Schleife beendet ist, wird der Programm-Anwender aufgefordert, eine Taste zu drücken. Dazu kennt Basic 2.0 einen oft benutzen Basic-Ausruck: POKE 198,0: WAIT 198,1. Adresse 198 gehört zum Tastaturpuffer des C 64 und speichert die Anzahl der gedrückten Tasten. Da diese Speicherstelle den Tastenanschlag aber erst erwartet, muß der Inhalt gelöscht werden (0): Der Computer befindet sich in einer Warteschleife, bis WAIT die Bedingung 1 vorfindet: Taste gedrückt!. Jetzt arbeitet der Computer den nächsten Befehl ab, also END.

Kurzinfo: fehlermeldungen

Dieses Programm hält sich ans Prinzip von »basic-befehle«:

Zeile 9: Bildschirm löschen, Kleinschrift aktivieren und Variablen vorbesetzen. TB erhält hier dem Wert 1 (zweite Spalte!).

Zeile 10: Die Schleife berücksichtigt nun die ROM-Adressen 41374 bis 41757: Hier hat der Basic-Interpreter die Texte zu den Fehlermeldungen abgelegt.

Zeile 28: ist identisch mit der Zeile in »basic-befehle«. Lediglich X wird hiier mit 20 multipliziert, um einen größeren Spaltenabstand bei der Bildschirmausgabe zu erhalten.

Zeilen 29 bis 50: identisch mit »basic-befehle«.

findet sich allerdings nicht mehr im Basic-ROM, sondern ist im Betriebssystem (Kernel) untergebracht: von Adresse 58402 bis 58539.

Befehle und Funktionen

Man teilt die Basic-Befehlswörter in Gruppen ein:

Kommandos: Sie gelten sowohl im Direkt- als auch im Programmodus (Beispiel: LOAD, SAVE).

Anweisungen: führt der Computer nur dann aus, wenn sie in einer Programmzeile definiert sind. Andernfalls erscheint die Fehlermeldung: Illegal direct (Beispiel: INPUT).

Funktionen: bezeichnen einen Wert, der durch die Art der Funktion und des Arguments (in Klammern) definiert ist (z.B. Quadratwurzel: SQR(49) = 7, FRE(0) usw.).

Operatoren: sind Rechenzeichen, die eine Beziehung zwischen Zahlen herstellen (z.B. +, -, / und *) oder zum Vergleich dienen (z.B. =, <, >, <>).

Sonder- und Grafikzeichen: In Verbindung mit Variablennamen ordnen bestimmte Zeichen die Variablenwerte deren Bedeutung zu: <$> registriert Zeichenketten (String-Variablen). <%> kann nur hinter numerischen Variablen stehen und legt fest, daß es sich um eine Integer-Variable ohne Nachkommastellen handelt (Beispiele: A$, A%). Andere Sonderzeichen fungieren lediglich als Satzanweisung (z.B. !, ?). <;> unterdrückt bei der PRINT-Ausgabe Leerzeilen und fügt die nächste Ausgabe direkt dahinter. <,> kann zwischen mehreren Variablen stehen, die der Computer der Reihe nach übernimmt (z.B. bei INPUT, READ, PRINT usw.). Unsere Tabelle 2 zeigt alle Befehlswörter des Basic 2.0 mit Erläuterung.

Tabelle 2. Die Befehle des Basic 2.0

Bezeichnung	Infos zur Bedienung
ASC("X")	verwandelt ein Zeichen (oder das erste einer Zeichenkette) in den ASCII-Codewert. Beispiel: PRINT ASC("A") ergibt »65«.
CHR$(X)	wandelt die ASCII-Codezahl X (0 bis 255) in das entsprechende Zeichen um. Die Zuordnung der ASCII-Codes zu den betreffenden Zeichen finden Sie im Handbuch des C 64. Beispiel: PRINT CHR$(65) bringt ein »A«.
CLOSE	schließt eine mit OPEN geöffnete Datei. Dazu genügt die Angabe der Dateinummer. Jede Datei muß unbedingt durch diese Anweisung geschlossen werden, sonst fehlt die Endmarkierung hinter dem letzten Datensatz. Das bringt Datenverlust. Nicht geschlossene Dateien erkennt man im Directory einer Diskette am »*«.
CLR	löscht alle Variablen und Array-Dimensionen: Numerische Variablen werden auf »0« gesetzt, String-Variable bekommen den Wert "" (leer) zugewiesen. Bei READ-DATA-Befehlen wird wie bei RESTORE der Zeiger auf die erste DATA-Zeile gelenkt. Das Programm im Speicher und der Zustand geöffneter Dateien bleiben davon unberührt.

CMD	ist die Abkürzung von »Change Main Device« und bewirkt, daß die folgende Datenübertragung auf die beim OPEN-Befehl angegebene Gerätenummer umgeleitet wird. CMD muß dieselbe Dateinummer wie OPEN besitzen. Alle weiteren Ausgaben (z.B. PRINT, LIST usw.) werden jetzt zum eingestellten Gerät geleitet.
	Beispiel: Ausgabe eines Basic-Listings auf dem Drucker
	`OPEN 4,4,0`
	`CMD 4`
	`LIST`
	CMD läßt sich auch mit einem Kommentar versehen, der dann ebenfalls mitgedruckt wird, z.B.:
	`CMD 4,"MEIN ERSTES LISTING"`
	Die Umstellung durch CMD wird mit PRINT# Dateinummer oder <RUN/STOP RESTORE> wieder rückgängig gemacht.

CONT	setzt ein mit STOP abgebrochenes Programm wieder in Gang: Alle Variablenwerte bleiben erhalten. Voraussetzung: Nach STOP durften keine Programmänderungen gemacht werden (z.B neue Basic-Zeilen dazuschreiben oder bestehende verbessern!). Dann erscheint die Fehlermeldung: »can't continue error«.
DATA	Damit kann man beliebig viele Zahlen und Strings in einem Programm fest speichern (nur begrenzt durch den Speicherplatz des Computers). Sie stehen in Programmzeilen hinter dem DATA-Befehl, getrennt durch Kommas. Strings müssen nicht in Anführungszeichen stehen, außer, sie enthalten Komma, Strichpunkt oder Grafikzeichen, die mit den Tasten <SHIFT> und <CBM> erzeugt werden.

DEF FN	Syntax-Vorschrift: DEF FN Name(Variable) = Formel. Komplizierte numerische Formeln lassen sich damit in Kurzform definieren. Der Name dieser Funktion beginnt jetzt mit FN. Der jeweilige Wert der numerischen Variablen in Klammern wird überall in der Formel eingesetzt, wo das Programm diese Variable findet. DEF FN darf man nicht im Direktmodus anwenden. Stringvariable sind nicht erlaubt.
	Beispiel: Die Adresse 49152 soll in High- und Lowbyte zerlegt werden.
	`10 DEF FN H(X) = INT(X/256)`
	`20 DEF FN L(X) = X-INT(X/256)*256`
	`30 AD = 49152`
	`40 PRINT FN L(AD), FN H(AD)`
	`Ausgabe: 0 (Lowbyte)  192 (Highbyte)`

DIM	reserviert einen Speicherbereich für eine per Index festgelegte Anzahl von Variablen desselben Namens: Der Unterschied besteht lediglich in der jeweiligen Indexnummer. Diesen Variablenspeicher nennt man Feld oder Array. Mit DIM lassen sich numerische und Stringvariablenfelder vorbereiten. Aber: ein Feld kann immer nur einen einzigen Variablentyp enthalten. Sie können mehrere Felder hinter DIM reservieren lassen: Die Variablennamen müssen durch Kommas getrennt sein. Felder können mehrere Dimensionen besitzen. Bei mehrdimensionalen Arrays werden pro Index 11 Byte im Basic-RAM reserviert: Vorsicht bei zu hohen DIM-Werten!
END	Dieser Befehl kann für sich allein stehen oder nach einer IF-THEN-Anweisung als Aktion. Das Programm wird sofort beendet. Es erscheint die READY-Meldung und der blinkende Cursor.
FOR-TO STEP-NEXT	Hinter FOR steht die Schleifenvariable mit der Zuweisung des Anfangswerts, hinter TO der Endwert. Mit STEP läßt sich die Schrittweite einstellen. Sie kann auch negativ oder ein Dezimalbruch sein. Schleifen, die rückwärts zählen, brauchen immer eine negative STEP-Anweisung: FOR I = 10 TO 0 STEP -2. Mehrere Schleifen lassen sich verschachteln. Die innere Schleife muß aber immer vor der äußeren abgearbeitet werden. Bei NEXT dürfen Sie die Angabe der Schleifenvariablen (z.B. NEXT A) weglassen, wenn dadurch keine Verwechslungen mit anderen Schleifen entstehen können. Folgen bei Schachtelschleifen mehrere NEXT-Befehle hintereinander, kann man hinter einer einzigen NEXT-Anweisung alle Schleifenvariablen angeben (durch Kommas getrennt!). Beispiel: NEXT A,B,C. Hier muß man ebenfalls die Reihenfolge »innen vor außen« einhalten!
GET	holt eine Zahl oder ein Zeichen aus dem Tastaturpuffer und weist es der Variablen zu, die hinter GET steht. Bei fehlerhaftem Variablentyp bricht das Programm mit einer Fehlermeldung ab. GET ist ungeduldig und wartet nicht. Ist der Puffer leer, weist er der Variablen den Wert »0« oder einen Leerstring zu. Daher läßt er sich nur in einer GOTO-Schleife verwenden. Beispiel: `10 GET A$: IF A$="" THEN 10` Im Gegensatz zu INPUT sind bei GET alle Zeichen erlaubt – falls es auf einen String wartet.
GET#	Wie der normale GET-Befehl liest GET# einzelne Zeichen aus einer mit OPEN geöffneten Datei. Im Gegensatz zu INPUT# kann die Anzahl der Zeichen beliebig gewählt werden. GET# braucht dieselbe Dateinummer wie OPEN und läßt sich nicht im Direktmodus verwenden.
GOSUB-RETURN	springt zu einem Unterprogramm, das mit der hinter GOSUB angegebenen Zeilennummer beginnt. Dann kehrt der Computer zur nächsten Anweisung zurück, die direkt hinter dem GOSUB-Aufruf steht. Unterprogramme lassen sich schachteln und können sich selbst aufrufen – allerdings höchstens 25mal. Grund: zuwenig Platz im Stapelspeicher, der Unterprogrammaufrufe registriert. Trifft ein Programm ohne vorheriges GOSUB auf eine RETURN-Anweisung, bricht es mit der entsprechenden Fehlermeldung ab: »return without gosub«.

GOTO

veranlaßt den Computer, sowohl im Direkt- als auch im Programmodus auf die hinter dem Befehlswort angegebene Zeilennummer zu springen, z.B. GOTO 1000. Es dürfen höhere oder niedigere Zeilennummern sein, durch Rücksprünge lassen sich Programmschleifen erzeugen. Fehlt die hinter GOTO eingetragene Zeilennummer, bricht das Programm mit der Fehlermeldung »undef'd statement error« ab.

IF - THEN

Nach IF folgt eine Prüfbedingung, nach THEN die Aktionsanweisung. Ist die Bedingung erfüllt, werden die restlichen Befehle in der Basic-Zeile ausgeführt, sonst macht das Programm bei der nächsten Zeile weiter. Mehrere IF-THEN-Befehle bilden, hintereinandergestellt, eine Mehrfachprüfung. Steht hinter THEN ein GOTO, können Sie einen der beiden Befehle weglassen. Beispiele: IF A = 1 THEN 100 bzw. IF A = 1 GOTO 100.

INPUT

funktioniert nicht im Direktmodus und darf nur in einem Basic-Programm eingesetzt werden. Stößt der Interpreter auf eine INPUT-Anweisung, gibt er ein Fragezeichen aus. Der Computer erwartet eine Eingabe von der Tastatur, die man mit <RETURN> abschließen muß. Je nach Variablentyp hinter INPUT (getrennt durch ein Semikolon), dürfen Sie Stringtexte oder beliebige Zahlen, auch mit Nachkommastellen, eingeben. Bei Strings kann Ihre Eingabe 78 Zeichen nicht überschreiten. Die Zeichen <,> und <:> sind nicht erlaubt. Stimmen Eingabe- und Variablentyp nicht überein, weist sie der INPUT-Befehl mit einer entsprechenden Meldung zurück. Per INPUT lassen sich mehrere Variablentypen hintereinander abfragen. Sie müssen durch ein Komma getrennt sein. Es ist möglich, vor die Abfrage per PRINT oder bei INPUT selbst einen Hinweiskommentar-String zu setzen, z.B.:

```
INPUT "KOMMENTAR";A$
```

Achten Sie darauf, daß der Kommentar-String nicht länger als 38 Zeichen ist!

INPUT#

liest Daten einer Datei, die mit OPEN geöffnet wurde. INPUT# verlangt die Angabe derselben Dateinummer (wie OPEN). Diese Anweisung läßt sich nur im Programmmodus anwenden. Es werden ganze Datengruppen in den Computer geholt (im Gegensatz zu GET#, das nur Zeichen für Zeichen liest) – allerdings darf kein String länger als 80 Zeichen sein. Hinter INPUT# können mehrere, durch Kommas getrennte Variable stehen, z.B. INPUT#2,A,B$,C,D.

INT(A)

wandelt den Dezimalbruch A in eine ganze Zahl (Integer) um. A kann nicht nur ein Dezimalbruch, sondern auch eine mathematische Formel bzw. ein komplizierter Rechenausdruck sein. Achtung: Lassen Sie nie die Klammern weg. Ein Beispiel: Die Formel

```
A = INT(RND(0)*X)+Y
```

erzeugt ganze Zahlen innerhalb des Zahlenbereichs Y bis Y + X - 1.

LEFT$(X$,A)

schneidet vom String X$, ganz links beim ersten Zeichen beginnend, A Zeichen ab und bildet daraus einen neuen String. Beispiel:

```
X$ = "COMPUTER"
PRINT LEFT$(X$,3)
Ausgabe: COM
```

LEN("string")

gibt die gesamte Länge eines Strings inklusive Leer- und Steuerzeichen als Zahl aus. In der Klammer darf ein String als Zeichenkette in Anführungszeichen oder als Variable stehen, z.b. LEN(X$).

LET

weist einer Variablen einen Wert zu: LET A = 10. Beim Basic 2.0 des C64 können Sie LET problemlos weglassen, die Anweisung »A = 10« bewirkt dasselbe.

LIST

gibt den Inhalt des Basic-Programmspeichers auf dem Bildschirm aus. Durch Eingabe der Zeilenummer oder eines solchen Bereichs können Programmteile auf den Monitor gebracht werden, z.B. LIST 10, LIST 100 - 300 usw.. Der Befehl LIST löscht kein Basic-Programm!

MID$(X$,B,A)

extrahiert aus dem String X$, von links beginnend, ab dem Zeichen mit der Nummer B insgesamt A Zeichen. Beispiel:

```
X$ = "INTERFACE"
PRINT MID$(X$,5,3)
Ausgabe: RFA
```

Wenn man die zweite Zahl A wegläßt, z.B. MID$(X$,5), wirkt dieser Befehl wie RIGHT$(X$,A).

NEW

macht den Programmspeicher frei zur Eingabe eines neuen Basic-Programms. Genauer: Es löscht die »Zeilen-Linker« (Adressen 2049/2050), die auf den Beginn der nächsten Programmzeile weisen und setzt die Zeiger fürs Ende des Basic-Programms zurück. Ab Adresse 2051 befindet sich Ihr altes Basic-Programm unversehrt im Speicher. Es läßt sich aber mit Tricks wieder zurückholen.

ON GOSUB/ON GOTO

Syntax-Vorschrift: ON Variable GOSUB/GOTO Zeile A, Zeile B, Zeile C usw. Ganzzahlige Variablenwerte legen fest, wohin das Programm springen soll: ON beginnt immer bei »1« zu zählen: 1 entspricht Zeilennummer A, 2 dagegen Zeilennummer B usw. Bei kleineren oder größeren Variablenwerten springt ON-GOSUB/ON-GOTO zur nächsten Basic-Zeile, ohne irgendein Unterprogramm aufzurufen. Haben Sie sich für ON-GOSUB entschieden, muß das aufgerufene Unterprogramm mit einer RETURN-Anweisung enden.

OPEN

öffnet eine Datei zum Schreiben oder Lesen. Hinter OPEN stehen bis zu sechs weitere Angaben, die man durch Kommas voneinander trennen muß: OPEN Dateinummer, Gerätenummer, Sekundäradresse, "Dateiname, Typ, Modus".

PEEK

Syntax-Vorschrift: PRINT PEEK(Adresse). »Adresse« muß immer in Klammern stehen: Die Anweisung liest den Inhalt dieser Speicherstelle. Erlaubt sind Zahlen zwischen 0 und 65535 (sonst erscheint ein »illegal quantity error«).

POKE

Syntax-Vorschrift: POKE Adresse, Wert. Adresse und Wert müssen durch ein Komma getrennt sein. »Wert« wird in »Adresse« eingetragen. Der alte Inhalt von »Adresse« wird dabei überschrieben. Erlaubte Zahlen für »Wert« sind 0 bis 255, für »Adresse« 0 bis 65535. Bei Überschreitung dieser Zahlen erscheint die Fehlermeldung »illegal quantity error«.

POS(0)

liefert die aktuelle Cursor-Position innerhalb einer logischen Bildschirmzeile. Dadurch können nur Werte zwischen 0 und 79 erscheinen. POS zeigt immer die Cursor-Position hinter dem zuletzt ausgegebenen Zeichen:

```
PRINTTAB(10)"HALLO";:PRINT POS(0)
Ausgabe: HALLO 15
```

Das Wort »Hallo« wurde ab der 10. Bildschirmspalte ausgegeben, fünf Zeichen (15) dahinter befindet sich jetzt der Cursor. In den Klammern hinter POS muß lediglich ein Dummy-Argument stehen, z.B. »0«.

PRINT

Der Computer gibt alles, was dahinter als Zeichenkette in Anführungszeichen, String- oder numerische Variable steht, auf dem Bildschirm aus. Das können auch Ergebnisse von Rechenanweisungen sein (z.B. PRINT 45 * 5, PRINT 49/7, PRINT 56 + 31 usw.). Mehrere Wörter oder Zahlenwerte lassen sich mit einem einzigen PRINT-Befehl ausgeben. Nur: Sie müssen durch ein Semikolon oder Komma voneinander getrennt sein. Der Strichpunkt erzeugt einen Abstand von zwei Leerzeichen, das Komma verschiebt die Ausgabe aufs nächste Bildschirmviertel. Zahlen oder Buchstaben, die ohne Anführungsstriche hinter PRINT stehen, werden immer als Variablen interpretiert. Alle Symbole der Tastatur, auch die Grafikzeichen, dürfen zur Bildung einer Zeichenkette (String) verwendet werden.

PRINT#

sendet Daten zur Floppystation oder zum Drucker. Pro PRINT#-Anweisung lassen sich maximal 255 Zeichen abschicken. PRINT# verlangt die gleiche Dateinummer wie OPEN. Es lassen sich numerische oder Stringvariable (auch DIM-Arrays) übertragen. Wichtig: Die einzelnen Variableninhalte müssen durch ein »Carriage Return« (CHR$(13) voneinander getrennt sein.

READ

ist untrennbar mit der DATA-Anweisung verbunden: READ liest der Reihe nach die Daten aus den Basic-Zeilen, die mit DATA beginnen, und legt sie im Variablenspeicher ab. Datentyp bei DATA und Variablentyp bei READ müssen unbeingt übereinstimmen.

REM

Damit kann man Kommentartexte in einer Basic-Zeile unterbringen. Eine REM-Zeile beeinflußt den Programmablauf nicht. Lediglich bei der Ausgabe mit LIST auf Bildschirm oder Drucker geben die Texte in REM-Zeilen aufschlußreiche Hinweise zur Programmfunktion.

RESTORE

hat nur Sinn in Verbindung mit READ und DATA. Falls die selben DATAs vom Programm nochmals gelesen werden sollen, muß man mit RESTORE den Zeiger für READ wieder auf die erste DATA-Zeile zurücksetzen.

RIGHT$(X$,A)

schneidet vom String X$, ganz rechts beim letzten Zeichen beginnend, A Zeichen ab und bildet einen neuen String. Beispiel:

```
X$ = "BARGELD"
PRINT RIGHT$(X$,4)
Ausgabe: GELD
```

RND(X)

setzt man für X die Zahl »0« oder irgendeinen positiven Wert ein, erzeugt dieser Befehl eine neunstellige Nachkommazahl zwischen 0 und 1. Sie ist quasi zufällig.

RUN

startet ein Basic-Programm im Speicher. Der Befehl darf beliebig oft wiederholt werden. Er löscht kein Programm. Folgt auf RUN eine Zeilennummer, beginnt der Programmablauf erst an dieser Stelle.

SPC(X) verarbeitet in den Klammern eine Zahl von 0 bis 255. Man verwendet diese Anweisung in Verbindung mit dem PRINT-Befehl. Die Zahl oder der String, der mit PRINT auf den Bildschirm gebracht werden soll, erscheint exakt nach den mit SPC(X) definierten Leerstellen, gerechnet ab der aktuellen Cursor-Position. Davorstehende Zeichen werden nicht gelöscht. Beispiel: PRINT SPC(80) ″TEST″ schreibt das Wort TEST zwei Zeilen (2 x 40) tiefer auf den Bildschirm.

STOP bricht ein laufendes Programm ab. Im Unterschied zum END-Befehl meldet sich der Computer mit »break in (Zeilennummer)«.

STR$(X) bildet aus dem Wert der Zahl X eine Zeichenkette, die man dann allerdings nicht mehr zu Rechenoperationen verwenden kann. Die Leerstelle fürs Vorzeichen wird beibehalten.

SYS Syntax-Vorschrift: SYS Adresse. Die Anweisung startet ein Maschinenprogramm, das bei »Adresse« beginnt. Sind Parameter fürs X-, Y-, P-Register oder den Akkumulator erforderlich, lassen sich die entsprechenden Werte in die Speicherzellen 780 bis 783 eintragen.

TAB(X) wirkt, wie SPC(X), nur in Verbindung mit dem PRINT-Befehl: Die Ausgabe eines gewünschten Strings oder einer Zahl geschieht exakt an der durch X festgelegten Position. X kann Werte zwischen 0 und 255 annehmen.

**TI, TI$** sind Variablennamen, die das Betriebssystem zur Ausgabe und Einstellung der internen Uhr verwendet. TI gibt den aktuellen Stand des Uhrzählers in Zehntelsekunden aus, TI$ wandelt diesen Wert in eine verständliche Anzeige um: Stunden, Minuten und Sekunden. Außerdem läßt sich durch eine neue Definition der sechsstelligen Zeichenkette TI$ die interne Uhr nach Wunsch einstellen. Beispiel: TI$ = ″103045″ setzt die C-64-Uhr auf 10 Uhr 30 und 45 Sekunden.

USR(X) arbeitet wie SYS (Aufruf einer Maschinenspracheroutine aus einem Basic-Programm). Man kann diesem Befehl einen numerischen Ausdruck als Argument mitgeben.

VAL(A$) liefert den numerischen Wert von A$. Der String darf als Zeichenkette (dann in Anführungszeichen) oder als Variable erscheinen. Die VAL-Funktion sucht den String A$ Zeichen für Zeichen nach einer Zahl ab. Beim ersten Zeichen, das keine Zahl mehr ist, wird abgebrochen. Falls der String zu Beginn überhaupt keine Zahl enthält, gibt VAL den Wert »0« aus.

WAIT Syntax-Vorschrift: WAIT Adresse,N1,N2. Dieser Befehl unterbricht ein Programm so lange, bis sich der Inhalt der »Adresse« von »0« unterscheidet. Diese Änderung wird durch die beiden numerischen Variablen N1 und N2 erzeugt: N2 verknüpft der Basic-Interpreter mit der Boole'schen Funktion EXOR, das Ergebnis anschließend per AND mit N1. Das Programm bleibt so lange unterbrochen, wie das Gesamtergebnis der beiden Verknüpfungen anders als »0« ist: Sonst macht das Programm mit dem nächsten Befehl weiter.

Veränderliche Werte: Variablen

Kein Basic-Programmierer kommt ohne sie aus: Variablen, die aus reinen Zeichenketten oder Zahlenwerten bestehen können. Sie lassen sich sowohl im Direktmodus als auch im Programm definieren. Es sind symbolische Platzhalter mit charakteristischem Namen. Gegenüber anderen Basic-Versionen besitzt das Basic 2.0 des C 64 zwei gravierende Nachteile: Auch, wenn sie längere Variablennamen eintippen (das ist ohne weiteres möglich), akzeptiert der Basic-Interpreter nur die ersten beiden Zeichen! Außerdem darf ein Variablenname pro Typ (String oder numerisch) nur einmal im Programm eingesetzt werden: aufpassen! Es geht zwar, z.B. den String C$ und die numerische Variable C im selben Programm zu definieren, aber man darf diese Namen nicht mehr an anderer Stelle (z.B. in einem Unterprogramm) und mit anderen Werten verwenden: Dann produziert das Basic-Programm mit Sicherheit nur Kauderwelsch. Trotzdem: kein Grund, zu verzweifeln! Variablennamen können Kombinationen aus zwei Buchstaben sowie eines Buchstabens und einer Zahl zwischen 0 und 9 bilden. Da gibt's eine Menge Möglichkeiten (z.B. AA$, B3$, C4$, D9, X1 usw.). Eines muß man dabei unbedingt beachten: Variablennamen dürfen nie mit einer **Zahl** beginnen (falsch ist: 3A, 6C, 7G$ usw.).

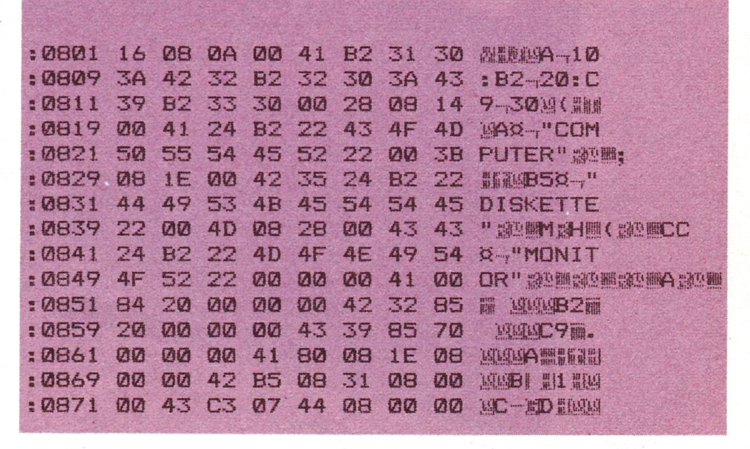
Variablennamen und deren Werte bewahrt der Computer in speziellen Speicherbereichen: unmittelbar hinter dem Ende des Basic-Programms (numerische und String-Variablen mit den Adressen, wo deren Text steht). Der Speicherbereich für numerische Variablen und String-Namen bewegt sich aufwärts, Richtung Basic-RAM-Ende. Denken Sie immer daran, wenn Sie umfangreiche Basic-Programme (über 120 Blocks) schreiben und dabei viele Variablen verwenden! Bei den Urversionen des C 64 wurden die Texte der im Programm angegebenen String-Variablen nochmals am obersten Ende des Basic-RAM angesiedelt und bewegten sich in Richtung Ende des Basic-Programms. Dort lagen aber die numerischen Variablen mit dem Trend, sich nach oben zu verbreiten. Wenn sich beide Variablentypen in der Mitte trafen, gab's einen »Out of memory Error« (kein Speicherplatz mehr frei!). Das hat Commodore in neuen C-64-Modellen und im C-64-Modus des C 128 abgestellt: Wurden im Basic-Programm bereits String-Variablen und deren Texte definiert, greift der Computer exakt auf diese Speicheradressen innerhalb des Basic-Programms zu und spart damit wertvollen Speicherplatz.

Tippen Sie folgende Programmzeilen ab:

```
10 a=10: b2=20: c9=30
20 a$="computer"
30 b5$="diskette"
40 cc$="monitor"
```

Nach dem Start mit RUN ergibt sich diese Verteilung des Programm- und Variablenspeichers (**Abb. 3**):

Das Basic-Programm liegt im Bereich ab Adresse 2049 ($0801 hexadezimal) bis 2126 ($084E). Unmittelbar dahinter (ab 2127) findet man die erste numerische Variable, die im Programm festgelegt wurde: A (als Hexadezimal-Byte 41). Das Null-Byte dahinter weist darauf hin, daß der Name nur aus einem Zeichen besteht und numerisch ist. Der Variablenwert (10) ist nicht auf Anhieb zu erkennen: Dazu benutzt der Computer die Methode der Fließkommadarstellung (hexadezimal 84 20). Das gilt auch für »B2« ab Adresse 2134 ($0856): Der Variablenname umfaßt jetzt zwei Hexadezimal-Byte (»42« für B, »32« für die Zahl 2), der Fließkommawert für »20« lautet hier »85 20«. Die letzte numerische Variable, C9, wird als »43 39« dargestellt und bekommt die Fließkommazahl »85 70« zugewiesen. Ab Speicherstelle 2148 beginnen die String-Variablen: »A$« ist mit »41 80« gekennzeichnet (die Hexadezimalzahl »80« ist das zweite Variablen-Byte, um den Wert »128« erhöht. Sie macht den Basic-Interpreter



```
:0801 16 08 0A 00 41 B2 31 30
:0809 3A 42 32 B2 32 30 3A 43  :B2 20:C
:0811 39 B2 33 30 00 28 08 14  9 30 (
:0819 00 41 24 B2 22 43 4F 4D  A¤ "COM
:0821 50 55 54 45 52 22 00 3B  PUTER" ;
:0829 08 1E 00 42 35 24 B2 22  B5¤ "
:0831 44 49 53 4B 45 54 54 45  DISKETTE
:0839 22 00 4D 08 28 00 43 43  " M H ( CC
:0841 24 B2 22 4D 4F 4E 49 54  ¤ "MONIT
:0849 4F 52 22 00 00 00 41 00  OR" A
:0851 84 20 00 00 00 42 32 85  B2
:0859 20 00 00 00 43 39 85 70  C9 .
:0861 00 00 00 41 80 08 1E 08  A
:0869 00 00 42 B5 08 31 08 00  B 1
:0871 00 43 C3 07 44 08 00 00  C D
```

[3] Die Variablen liegen hinter dem Basic-Programm

Bücher und Software

Mehr Fun

Withöft/Draheim

64'er – Großer Einsteiger-Kurs

Leichtverständlicher Einführungskurs in die Welt des C64. Die Autoren dokumentieren mit diesem Buch ihre lehrreichen Erfahrungen, Erfolge und auch Fehlschläge. So entstand ein Buch, das den Computer-Neuling Schritt für Schritt durch die Welt des C64 führt. Angefangen vom Auspacken und Anschließen des C64, über Basic-Programmierung bis zu PEEK- und POKE-Befehlen oder der Sprite-Programmierung wird das für den Einsteiger Wissenswerte über den C64 behandelt. Mit allen großen Beispielprogrammen auf der beiliegenden Diskette.
1988, 236 S., inkl. Diskette
ISBN 3-89090-668-0, DM 29,90

Über 50.000 verkaufte Exemplare!

Markt&Technik

Das Buch zum erfolgreichen 64'er-Einsteigerkurs: Henning packt aus

Henning Withöft
Andrew Draheim

64'er
Großer Einsteiger-Kurs

Leichtverständliche Einführung in die Welt des C64: Monitor ★ Diskettenstation ★ Basic-Programmierung: Grafik ★ Spiele ★ Musik

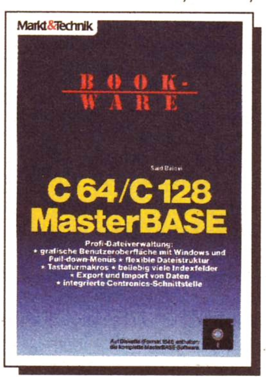
Auf 5¼"-Diskette im Format 1541/70/71 enthalten: Alle großen Beispielprogramme.



Software zum Buchpreis: Profi-Textverarbeitung mit komfortabler Druckeranpassung, Spellchecker, Zeichensatz-Editor und Adreßverwaltung. Außerdem: individuelle Farbgebung, komprimiertes Dateiformat, freie Tastenbelegung, Silbentrennung und … und … und …
1988, 201 S., inkl. Diskette
ISBN 3-89090-527-7, DM 59,–*



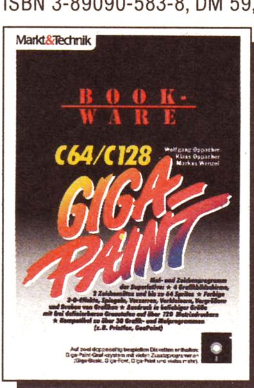
C64/C128-Software zum Buchpreis: eines der professionellsten 3D-CAD-Programme, das Grafiken von starker Räumlichkeit und Faszination erstellt. Ermöglicht ruckfreie Filme mit bis zu 24 Bildern pro Sekunde. Die Bildkonstruktion erfolgt mit dem Joystick.
1986, 183 S., inkl. Disketten
ISBN 3-89090-409-2, DM 49,–*



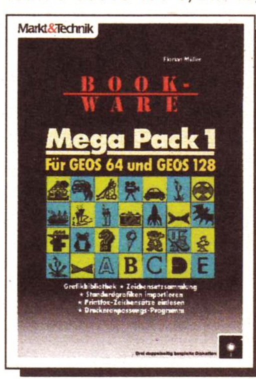
Software zum Buchpreis. Profi-Datenbank: grafische Benutzeroberfläche mit Windows und Pull-Down-Menüs, flexible Dateistruktur, Tastaturmakros, beliebig viele Indexfelder, Export und Import von Daten, integrierte Centronics-Schnittstelle.
1988, 155 S., inkl. Diskette
ISBN 3-89090-583-8, DM 59,–*



CAD-Programm für technisches Zeichnen, Malen in Farbe, Glückwunschkarten und sogar kleine Trickfilme. Mit 234 Seiten Dokumentation und vielen Anwendungsbeispielen. Außerdem ein komfortables Druckeranpassungsprogramm.
1985, 234 S., inkl. Diskette
ISBN 3-89090-136-0, DM 48,–



Super-Malprogramm zum Buchpreis: 4 Grafikbildschirme, 2 Zeichensätze und bis zu 64 Sprites; farbige 3D-Effekte; Grafiken spiegeln, verzerren und drehen; Ausdruck in beliebiger Größe mit frei definierbaren Graustufen. Zahlreiche Zusatzprogramme.
1988, 261 S., inkl. 2 Disketten
ISBN 3-89090-619-2, DM 59,–*



Software zum Buchpreis. Die nützliche Ergänzung für GEOS 64 und GEOS 128: Grafikbibliothek, Zeichensatzsammlung, Standardgrafiken importieren, Printfox-Zeichensätze einlesen und Druckeranpassungsprogramm.
1989, 160 S., inkl. 3 Disketten
ISBN 3-89090-772-5, DM 59,–*

für den C 64

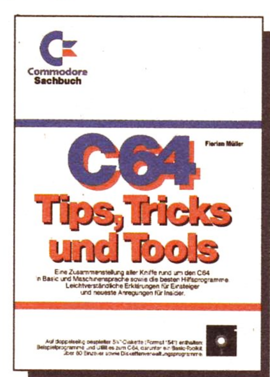
für Fans!



Die Fortsetzung von Mega Pack 1: insgesamt 1 MByte auf 3 Disketten mit mehr als 500 DISKART-Grafiken im GeoPaint-Format, 33 GEOS-Zeichensätzen und prima Grafik-, Disk- und Druck-Utilities. Die Dateien sind mit allen deutschen GEOS-Versionen verwendbar.
1989, 177 S., inkl. 3 Disketten
ISBN 3-89090-350-9, DM 59,–*



Eine Kombination aus Assembler-Kurs im Buch und Software auf Diskette: komplettes Entwicklungspaket mit Makroassembler, Maschinensprachenmonitor, Reassembler, Einzelschrittsimulator.
Die Entwicklung von eigenen Assemblerprogrammen wird so zu einer leichten Übung.
1988, 314 S., inkl. Diskette
ISBN 3-89090-571-4, DM 59,–



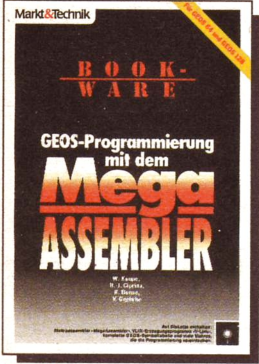
Eine Zusammenstellung aller Kniffe rund um den C64 in Basic und Maschinensprache. Leichtverständliche Erklärungen für Einsteiger und neueste Anregungen für Insider. Mit Beispielprogrammen und besten Utilities auf Diskette. Nach der Lektüre sind Sie der Profi auf dem C 64.
1988, 439 S., inkl. Diskette
ISBN 3-89090-499-8, DM 59,–



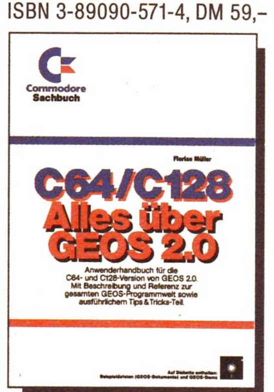
64'er Spielesammlung, Band 1: ★ Balliard ★ Maze ★ The Way ★ Schiffeversenken ★ Vager 3 ★ Firebug ★ Pirat ★ Handel ★ Wirtschaftsmanager ★ Börse ★ Vier gewinnt ★ Vier in Vier ★ Brainstorm ★ Schach ★ Zauberwürfel.
Die Entführung in eine abenteuerliche Spielewelt.
1987, 115 S., inkl. Diskette
ISBN 3-89090-429-7, DM 39,–*



64'er Spielesammlung, Band 2: ★ Golf ★ Billard ★ Tonti ★ Zauberschloß ★ Freiheit ★ Steel Slab ★ Space Invader ★ Apocalypse Now ★ Block Out ★ Aquator ★ Tödliches Dioxin ★ Libra ★ Dasher ★ Bundesliga-Manager.
Für alle, die vom Spielen nicht genug bekommen.
1987, 98 S., inkl. Diskette
ISBN 3-89090-428-9, DM 39,–*



Der ideale Einstieg in die Assembler-Programmierung unter GEOS. Damit kann jeder C64/C128-Benutzer eigene GEOS-Anwendungen, Desk-Accessories und VLIR-Programme schreiben. Mit allen Routinen und Systemvariablen zum Nachschlagen.
1990, 432 S., inkl. Diskette
ISBN 3-89090-247-2, DM 89,–*



Anwenderhandbuch für GEOS 2.0 (C64/C128). Mit Beschreibung und Referenz zur gesamten GEOS-Programmwelt sowie vielen Tips & Tricks.
Mit GEOS-Demo auf Diskette.
1989, 432 S., inkl. Diskette
ISBN 3-89090-808-X, DM 59,–



Eine leichtverständliche Einführung in die digitale Schaltungstechnik. Mit Platinenlayouts und Selbstbauanleitungen für Sprachausgabebaustein, 128-Kbyte-EPROM-Karte u. a. Mit Treibersoftware auf Diskette.
1987, 294 S., inkl. Diskette
ISBN 3-89090-389-4, DM 49,–



64'er Spielesammlung, Band 3: ★ Arabian Treasurehunt ★ Block'n'Bubble ★ Robo's Revenge ★ Race of the Bones ★ Quadranoid ★ Future Race ★ Risiko ★ Copter Flight ★ Asteroids 64 ★ Verminator ★ Der kleine Hobbit ★ Odyssey.
1988, 103 S., inkl. Diskette
ISBN 3-89090-596-X, DM 39,–*



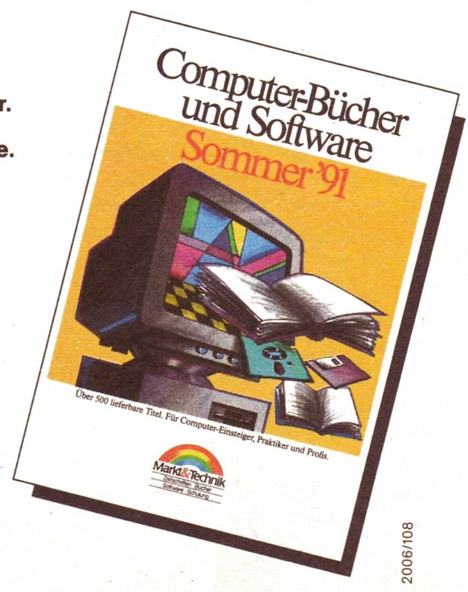
64'er Spielesammlung, Band 4: ★ Samurai: Kampfspiel mit Spitzengrafik ★ Omidar: mit einem Gorilla auf der Flucht ★ Bomb Runner: Minensuchkommando ★ Crillion: Mischung aus Boulder Dash und Arkanoid.
1988, 80 S., inkl. Diskette
ISBN 3-89090-703-2, DM 39,–*

* Unverbindliche Preisempfehlung



Spiele programmieren und dabei alles über Basic lernen. Die im Buch beschriebenen 21 Spiele sind als Listings abgedruckt und werden auf der beiliegenden Diskette gleich mitgeliefert.
1989, 209 S., inkl. Diskette
ISBN 3-89090-701-6, DM 39,–

Markt&Technik-Bücher und -Software gibt's überall im Fachhandel und bei Ihrem Buchhändler. Fragen Sie auch nach dem neuen Gesamtverzeichnis mit über 500 aktuellen Computer-Büchern und Software.





Zeitschriften · Bücher
Software · Schulung

2006/108

darauf aufmerksam, daß es sich jetzt um eine Zeichenketten-Variable handelt), dahinter steht die Länge des String-Textes (08) und die Adresse in Low-Highbyte-Darstellung, ab der er im C64 gespeichert ist: $081E. Dies gilt auch für die restlichen String-Variablen B5$ und CC$: Das Wort »Diskette« beginnt bei 2097, »Monitor« bei 2116. Wenn künftig im Basic-Programm oder Direktmodus diese Variablen aufgerufen werden, holt sie sich der C64 aus diesen festgelegten Speicherbereichen und bringt ihren Wert bzw. Text auf den Bildschirm.

Man unterscheidet drei Variablentypen:

Real: Das sind numerische Variablen. Sie besitzen Nachkommastellen, also Zahlenbrüche hinter dem Dezimalpunkt. Man verwendet sie speziell für Berechnungen. Sie tragen kein Sonderzeichen hinter dem Variablennamen.

Integer: betrifft nur ganzzahlige Werte ohne Stellen hinter dem Komma. Man muß sie mit dem Prozentzeichen <%> ausstatten. Vorteil: Sie bringen eine leicht lesbare Zahlendarstellung: High- und Low-Byte als Hexadezimalzahlen (bei Fließkommazahlen ist es nicht einfach, auf Anhieb deren Wert zu erkennen!). Nachteil: Es sind nur Werte zwischen -32768 bis +32767 möglich.

Zeichenketten (Strings): dürfen alle Buchstaben, Ziffern, Satz-, Sonder- und Grafikzeichen der Tastatur enthalten (außer Anführungsstrichen, die fixieren nämlich Anfang und Ende eines Strings!) und müssen nach dem Variablennamen ein Dollarzeichen <$> haben.

Real-, Integer- und String-Variablen verbrauchen exakt den gleichen Speicherplatz: 7 Byte (zwei für den Namen, fünf für die Zahlenwerte bzw. die String-Adresse).

Verbotene Variablen

Hatten Sie schon »Syntax Errors« in offensichtlich korrekten Basic-Zeilen? Dann waren bestimmt die verbotenen Variablen schuld daran! Tippen Sie beispielsweise folgendes Listing ein:

```
10 schiff$="titanic"
20 print schiff$
```

Nach RUN erscheint eine Fehlermeldung. Bei der Syntaxprüfung hat der C64 festgestellt, daß in der Variablen SCHIFF$ ein Basic-Befehlswort enthalten ist: IF. Das irritiert den Basic-Interpreter, der den Befehl in die Abkürzung (Tokens) umwandelt – der übrige Text der Basic-Zeile paßt syntaktisch nicht dazu. Die Liste solcher Stolpervariablen läßt sich beliebig fortführen: TANNE$ (enthält TAN), LAND$ (AND), ORT$ (OR), START$ (ST), TITEL$ und TICK$ (TI). Schade, denn auf solche selbsterklärenden Variablennamen muß man verzichten und auf weniger markante ausweichen. Wenn Sie jedoch einen kleinen, aber um so effektiveren Trick benutzen, können Sie den Interpreter übers Ohr hauen und ihm sogar verbotene Variablen unterjubeln: durch Verwendung des Grafikzeichens <SHIFT J> innerhalb des Basic-Befehlswortes im Variablennamen. Das folgende Programmbeispiel funktioniert einwandfrei, obwohl der Variablenname das Basic-Wort AND enthält (geben Sie <shift j> nicht als Text, sondern als entsprechendes Tastaturzeichen ein):

```
10 la<shift j>nd$=
"italien"
20 print la<shift j>nd$
```

Die Zeichenfolge, die der Basic-Interpreter als Befehlswort akzeptieren könnte, muß lediglich durch das Grafikzeichen getrennt werden. Wenn Sie so verfahren, sollten Sie aber darauf achten, daß der entsprechende Variablenname mit dem Grafikzeichen im gesamten Programm gleichlautend geschrieben wird!

Trotz dieses Tricks gilt weiterhin, daß der Computer nach erfolgreicher Interpretation jeden Variablennamen lediglich mit den ersten beiden Zeichen speichert. Falls Sie längere Variablenbezeichnungen benutzen möchten, müssen Sie folgendes beachten: TASTEN$ und TABELLE$ sind für den C64 ein- und dieselbe Variable: TA$.

Variablenfelder

Fortgeschrittene Programmierer speziell geben sich mit einfachen Variablen selten ab: Sie bevorzugen die indizierten oder Feldvariablen (Arrays). Egal, welchem Typ sie angehören: Ihr Name besteht stets aus der Variablenbezeichnung und mindestens einer Indexziffer in Klammern. Ihr großer Vorteil liegt darin, daß nur ein Name für gleichartige Variablen verwendet werden muß. Durch die Indextiefe (ein-, zwei-, dreidimensional usw.) ergeben sich unterschiedliche Variablenwerte, die sich nicht gegenseitig überschneiden oder stören. Die Menge und Größe solcher Indexfelder ist vor allem durch den Speicherplatz begrenzt: Sie müssen nämlich vorbesetzt werden.

DIM-Anweisung

Ohne Dimensionierung verkraftet der C64 nur zehn Inidizes pro Variable: In den meisten Fällen ist das zu wenig. DIM darf nur einmal im Programm ausgeführt werden und sollte daher gleich zu Programmbeginn oder in einer Unterroutine stehen, die nach dem Programmstart ebenfalls nur einmal aufgerufen wird (sonst erhalten Sie die Fehlermeldung: Redim'd Array). Die kleinste Indexzahl ist immer »0« (negative Zahlen oder Nachkommastellen sind nicht erlaubt). Die DIM-Anweisung läßt sich im Direkt- und Programmodus anwenden und setzt zu Beginn alle spezifischen Feldelemente auf »0«. Benutzt man im Programmverlauf größere Indizes als mit DIM eingestellt, erscheint die Fehlermeldung: Bad Subscript. Achtung: Haben Sie mit DIM kleinere Werte als »10« eingestellt, wird auch nur die kleinere Dimensionierung akzeptiert! Der Freiraum nach dem Einschalten (automatisch zehn Dimensionierungen) gilt dann nicht mehr.

Bis zu 255 Felder lassen sich mit DIM einstellen. Jedes darf maximal 32767 Elemente enthalten. Allerdings: In der Praxis werden solche Monster-DIMs nie vorkommen!

Was nützen in Dimensionen unterteilte Datenfelder? Kurz gesagt: Damit lassen sich Angaben und Informationen zu einem bestimmten Objekt spezifizieren. Dazu ein Beispiel:

Sie sind Vorstand eines Vereins und möchten fünf Mitglieder erfassen. Brauchen Sie nur deren Namen, genügen eindimensionale Datenfelder, z.B.:

```
DIM NN$(5)
NN$(1)="MAIER Hans"
NN$(2)="HUBER Walter"
NN$(3)="SCHMIDT Georg"
NN$(4)="KRAUSE Werner"
NN$(5)="BERTHOLD Thomas"
```

Diese Liste ist recht kümmerlich: Wo bleiben die Adressen? Wichtig sind Straße, Wohnort und Telefonnummer. Mit dem Mitgliedsnamen sind das insgesamt vier Indizes. Legen Sie nun ein zweidimensionales Array an:

```
DIM NN$(5,4)
NN$(1,1)="MAIER Hans"
NN$(1,2)="Baumweg 7"
NN$(1,3)="8000 München 80"
NN$(1,4)="089/145678"
```

Der erste Datensatz »Maier Hans« besitzt nun **vier** Aussagen über das Mitglied, es wurden aber nur **zwei** Dimensionen verwendet. Lassen Sie sich von den Zahlenangaben »5,4« nicht verwirren!

Überwiegend zweidimensionale Datenfelder werden von Basic-Programmierern verwendet, da man durch die entsprechende Größe des zweiten Index zusätzliche Informationen zum Hauptbegriff sammeln kann. Folgende dreidimensionale DIM-Anweisung ist ebenfalls möglich und erzeugt für den Anwender den sichtbar selben Effekt: DIM NN$(5,2,2). Für welche Art Sie sich entscheiden, hängt nicht zuletzt davon ab, wie und mit welchen Sortierpro-

grammen die Daten aufbereitet werden sollen. Einen Nachteil von DIM wollen wir nicht verschweigen: Ein dimensioniertes Array, z.B. A$(1), verschlingt 17 Byte des wertvollen Speicherplatzes im C 64!

Als Demos zur DIM-Anweisung finden Sie auf unserer Diskette zum Sonderheft zwei Beispiele: **eindim** und **zweidim**.

Eingabesteuerung

Jeder Basic-Programmierer kennt sie, die:

INPUT-Anweisung

- ein Komfort des Betriebssystems, der Eingaben (Text oder Ziffern) bis zur Länge einer logischen Bildschirmzeile (80 Zeichen) erlaubt, allerdings nur im Programmodus. Der Computer stoppt den Ablauf, gibt ein Fragezeichen aus und wartet auf die Eingabe. Manko: Dieser Befehl hat einige Schwachstellen:

- Die Eingabezeile kann mit dem Cursor jederzeit verlassen werden, was Fehleingaben geradezu vorprogrammiert.

- Bestimmte Trennzeichen nimmt der INPUT-Modus nicht an oder unterbricht das Programm nach Abschluß der Eingabe mit <RETURN> durch eine Fehlermeldung: <:> (Doppelpunkt) und <,> (Komma). Diese Zeichen produzieren den Fehler »Extra ignored«.

- Wurde hinter der INPUT-Anweisung eine numerische Variable definiert (z.B. A), dürfen nur die Ziffern 0 bis 9 zur Eingabe verwendet werden (auch Realzahlen mit Nachkommastellen sind möglich!). Andernfalls meldet sich der Computer mit »Redo from Start« (nochmals von vorn!). Setzen Sie aber vor die Zeichenketteneingabe ein Anführungszeichen, übernimmt der Computer auch die verbotenen Zeichen <:> und <,>!

Werden String-Eingaben verlangt (z.B. A$), akzeptiert INPUT alle erlaubten Zeichen auf der Tastatur. Nachteil: Zahlen werden ebenfalls als Strings identifiziert und können nicht zu Berechnungen verwendet werden. Die INPUT-Anweisung darf auch Hinweistext enthalten, dahinter muß sich – abgetrennt durch das Semikolon <;> der Variablenausdruck des entsprechenden Typs befinden. Es dürfen auch indizierte Variablen sein. Einige Beispiele:

```
10 input"Geben Sie
eine Zahl    ein";z
```

Der C64 erwartet eine Real- oder Integerzahl, die in der Variablen Z gespeichert wird. Diese Zahl kann im weiteren Programmverlauf zu Berechnungen verwendet werden.

```
10 input"Ihr Text:";a$
```

Jetzt stehen alle Zeichen außer <:> und <,> auf der Tastatur für die Eingabe zur Verfügung. Wer ganz sicher gehen will, sollte die Zeichenkette mit dem Anführungszeichen beginnen lassen. Dann kann man sogar Farb- und Steuerzeichen (z.B Cursor-Bewegungen) in die spätere Textausgabe einbinden!

Eine weitere Variante des INPUT-Befehls bereitet dem Basic-Interpreter ebenfalls keine Schwierigkeiten: Mehrere INPUT-Fragen, auch verschiedenen Variablentyps, hintereinander:

```
10 input"Tag, Monat, Jahr,
   Name, Nr.";dd$,mm$,yy$,
   nm$,nr
```

Beachten Sie, daß zwar unmittelbar hinter der INPUT-Anweisung ein Semikolon stehen muß (vor DD$). Alle weiteren Variablen dürfen jedoch nur durchs **Komma** verbunden sein (sonst gibt's einen »Syntax error«). Ab der Frage zur zweiten Variable meldet sich der Computer immer mit zwei Fragezeichen.

Wer statt des Basic-Befehls INPUT lieber die Betriebssystemroutine SYS 65487 ($FFCF) benutzen will, sollte sich das Beispiel **input.sys** von der Diskette laden.

Vorteile von INPUT: Die Anweisung enthält nützliche Editorfunktionen (Löschen des Eingabetextes mit der DEL-Taste, Cursor-Bewegungen). Positiv eingestellte Anwender betrachten auch die sofortige Ausgabe der Fehlermeldungen bei der Eingabe (Extra ignored, Re-

Kurzinfo: eindim

Nach RUN erscheint in der obersten Bildschirmzeile die INPUT-Abfrage. Tippen Sie nun beliebige Namen ein. Falls Sie die Eingabe abbrechen möchten, drücken Sie lediglich die RETURN-Taste: Ihre bisherigen Eingaben erscheinen auf dem Bildschirm. Jeder weitere Tastendruck beendet das Beispielprogramm.

Zeile 20: Eine Schleife erzeugt den String LN$, der 38 Querstriche enthält.

Zeile 30: EG ist die maximale Zahl der INPUT-Eingaben. Man übernimmt sie in die DIM-Anweisung zu EG$.

Zeile 40: Der Obergriff des Datenfelds für die 20 Eingaben heißt »Name« und wird der Variablen DF$ zugewiesen.

Zeile 50: Hier beginnt die Eingabeschleife: Der INPUT-Befehl soll laut dem Wert von EG 20mal aufgerufen werden.

Zeile 52: ist der »Notausgang«. Ist der aktuelle Eingabestring leer (weil nur die RETURN-Taste gedrückt wurde), springt das Programm zu

Zeile 60: Der Bildschirm wird gelöscht, der Datenfeldname und der Strich LN $ tauchen auf der Bildfläche auf.

Zeile 70: Nun gibt der Computer innerhalb einer Schleife die bislang gesammelten Eingaben EG$(I) aus.

Zeile 1000: wartet auf einen Tastendruck und beendet das Programm.

Kurzinfo: zweidim

Prinzipiell geht das nächste Dateneingabeprogramm genauso vor, allerdings erläutern wir nur die Programmzeilen, die sich von **eindim** unterscheiden:

Zeile 30: EG legt in diesem Beispiel sieben Datensätze fest, die jeweils auf vier Eingaben bestehen (DF). Die DIM-Anweisung berücksichtigt jetzt zweidimensionale Arrays.

Zeile 40: Die Bezeichnungen der vier Datenfelder speichert der Computer in der eindimensionalen Feldvariablen DF$(DF): Name, Straße, Wohnort und Telefon.

Zeile 50: Pro Dimension muß die entsprechende Schleife aufgemacht werden: FOR I=1 TO 7 (EG), FOR J=1 TO 4 (DF). Die INPUT-Fragen erscheinen.

Zeile 59: Um bei der Bildschirm-Eingabe eine Abtrennung zwischen den Datensätzen zu erreichen, muß man zwischen die beiden NEXT-Befehle ein PRINT setzen.

Zeile 60: Die vier Datenfeldbezeichnungen bringt der Computer mit der Schleife K auf den Bildschirm und gibt durch **Zeile 70** die Datensätze mit jeweils vier Einträgen aus (Abb.4).

Rechnen Sie's nach: 28 Eingaben (7 x 4) sind bei diesem zweidimensionalen Array möglich.

Kurzinfo: input.sys

Dieses Demo aktiviert die Systemroutine $FFCF (SYS 65487), die man als Ersatz für INPUT verwenden kann. **input.sys** läßt sich als Unterroutine in eigenen Programmen verwenden (mit GOSUB anspringen und RETURN verlassen!). Ein großer Vorteil: Das störende Fragezeichen wird dabei nicht ausgegeben!

Zeile 10: Aufruf der Betriebssystemroutine. Sie komtrolliert den Inhalt von Adresse 780. Sie entspricht dem Register A (Akkumulator bei der C-64-Assemblerprogrammierung).

Zeile 20: überprüft, ob der empfangene Tastendruck die RETURN-Taste war (Akku = 13). Falls ja, verzweigt das Programm zu Zeile 50.

Zeile 25: Da die Eingabezeile nur der Wert einzelner Tasten registriert, muß die Variable EG$ um den Inhalt jedes weiteren Tastendrucks (PEEK(780)) ergänzt werden.

Zeile 40: Rücksprung zu Zeile 10, um den nächsten Tastenwert einzulesen.

Zeile 50: bringt die eingegebenen Zeichen auf den Bildschirm. Sie können im Gegensatz zu INPUT jedes beliebige Zeichen verwenden. Die Cursortasten sind zwar nicht gesperrt, aber das Verlassen der Eingabezeile bringt weder falsche Strings noch eine Fehlermeldung. Fehleingaben dürfen mit der DEL-Taste eliminiert werden.

do from Start) als Vorteil: Man erkennt sofort, was man falsch gemacht hat.

Auf geordnete und strukturierte Eingabemasken des Bildschirms wirken sich die Fehlermeldungen allerdings verheerend aus: Die Maske wird zerstört. Ebenso sollte eine Sperre bestehen, die verhindert, daß sich der Eingabe-Cursor aus Versehen selbständig macht und auf dem gesamten Bildschirm herumfuhrwerkt.

GET-Anweisung

Dieser Befehl liest jeweils ein Zeichen von der Tastatur. Der ASCII-Wert (CHR$) läßt sich in einer String-Variablen speichern. Je nach Eingabenlänge erhöht sich auch die Länge der Zeichenkette. Erlaubt sind alle Zeichen, Ziffern werden ebenfalls als Strings behandelt. Vorteil: Sie können die entsprechenden Editorfunktionen nach Ihren Wünschen programmieren: Sperren der Tasten <HOME>, <CRSR abwärts/aufwärts>, nur bestimmte Eingabezeichen zulassen usw. Die entsprechende Basic-Routine bringt das Programm **get.bas**. Es benötigt auf jeden Fall weniger Speicherplatz als **get.sys**, das die Betriebssystemroutine SYS 65508 ($FFE4) verwendet.

Ein gravierender Nachteil von GET: Man sieht keinen Eingabe-Cursor und weiß nie genau, in welcher Spalte er sich gerade befindet. Editorbewegungen in der Eingabezeile mit <CRSR links/rechts> werden damit zum Glücksspiel. Dazu bietet die Adresse 204 in der Zeropage des C64 eine Alternative, die einen blinkenden Eingabe-Cursor an der aktuellen Bildschirmposition erzeugt (POKE 204,0), doch ist es auch hier Glückssache, ob er nach getaner Arbeit (POKE 204,1) verschwindet oder nicht: Oft bleibt ein störender Cursor-Block auf dem Bildschirm zurück. Wenn Sie's trotzdem versuchen möchten, ergänzen Sie die Zeilen des Listings **get.sys**:

```
10 sys 65508: poke 204,0
40 poke 204,1: goto 10
```

Diese geänderte Version finden Sie unter dem Dateinamen **get.curs** ebenfalls auf der Diskette.

Basic fällt Entscheidungen

Die folgende Befehlsfolge bringt im Prinzip die gesamte Intelligenz von Basic 2.0 auf einen Nenner:

IF-THEN-Anweisung

Damit entscheidet das Programm, wie es sich bei bestimmten Bedingungen verhalten soll: Es kann ein Unterprogramm aufgerufen werden, der Computer in einer Warteschleife verharren oder das Programm beenden. Welche Kriterien der C64 befolgen soll, bestimmen Sie im Programm. Dazu müssen Sie die Vergleichsoperatoren von Basic 2.0 verwenden. Einige Beispiele:

```
IF X=Y THEN...
```

X besitzt exakt denselben Wert wie Y.

```
IF X<>Y THEN...
```

X unterscheidet sich von Y.

```
IF X>Y THEN...
```

X ist größer als Y.

```
IF X<Y THEN...
```

X ist kleiner als Y.

```
IF X>=Y THEN...
```

X hat einen größeren oder den gleichen Wert wie Y.

```
IF X<=Y THEN...
```

X ist kleiner oder zumindest gleich mit Y.

Es lassen sich numerische Variableninhalte, Zahlen, Rechenergebnisse und Strings vergleichen. Wenn das Vergleichsergebnis logisch wahr ist (—1), verzweigt der Computer zur programmierten Reaktionsroutine. Findet er keine Übereinstimmung (logisch falsch = 0), berücksichtigt der Basic-Interpreter nicht mehr die Anweisungen hinter THEN und geht zur nächsten Zeile.

IF-THEN-Anweisungen lassen sich auch verschachteln:

```
IF X>Y THEN IF A=B THEN...
```

Ziemlich verwirrend, nicht? Eleganter ist hier der Einsatz logischer Operatoren: AND, OR, NOT. Nachteil: Die Programmausführung dauert einiges länger als bei der unübersichtlichen Version!

Ihre Basic-Zeile könnte jetzt so aussehen:

```
IF X>Y AND A=B THEN...
```

AND-Verknüpfung: Das Programm verzweigt nur dann, wenn X größer als Y **und** A gleich B ist.

```
IF A<C OR X>Y THEN...
```

OR-Verknüpfung: Es genügt bereits, wenn nur eine der beiden gestellten Bedingungen erfüllt ist: Entweder ist A kleiner als C **oder** X größer als Y.

```
IF A=NOT C THEN...
```

NOT negiert jede positive Zahl und addiert -1. Trifft NOT auf Minuszahlen, werden diese positiv, sind aber um »1« reduziert: Das Vergleichsergebnis wird also umgedreht! Beispiele:
C = 17, NOT C = -18,
C = -17, NOT C = 16,
NOT 0 = -1, NOT -1 = 0.

Sicher reizvoll für Mathe-Freaks, kommt die NOT-Verknüpfung in Basic-Programmen sehr selten vor. Für Interessierte gibt's das Programmbeispiel **not** auf der Diskette zu unserem Sonderheft.

Ist eine IF-THEN-Bedingung erfüllt, muß bei Basic 2.0 die Reaktionsroutine in derselben Basic-Zeile, direkt hinter THEN stehen (oder ein entsprechender Sprungbefehl zu einem Unterprogramm mit GOSUB oder GOTO). Programmtechnisch ist zwar möglich, die Reaktion in die nächsten Zeilen zu setzen, die der IF-THEN-Abfrage folgen. Beachten Sie aber, daß dann dieser Programmabschnitt immer ausgeführt wird - egal, ob die geforderte Bedingung logisch wahr oder unwahr ist!

Wo soll's hingehen?

Man plaziert IF-THEN-Fragen in Basic-Program-

Kurzinfo: get.bas

Zeile 10: wartet auf die Eingabe von der Tastatur (T$).
Zeile 20: fragt unerlaubte Tastencodes ab (<CRSR aufwärts/abwärts>, <HOME>). Hat sich einer eingeschlichen, wird er nicht berücksichtigt (T$=0).
Zeile 30: War T$ die RETURN-Taste, springe zu Zeile 50!
Zeile 40: Gib das Zeichen der aktuellen Taste (T$) aus und füge es an die sich bereits in der Eingabezeile befindlichen an!
Zeile 42: EG$ wird mit T$ erweitert. Die gesamte Zeichenkette ist jetzt um dieses Zeichen länger.
Zeile 45: Rücksprung zu Zeile 10 (Aufruf der Systemroutine).
Zeile 50: wird angesprungen, wenn die RETURN-Taste gedrückt war: EG$ erscheint zum Test nochmals unter der Eingabe auf dem Bildschirm.

Kurzinfo: get.sys

Diese Variation aktiviert die Betriebssystemroutine von GET: $FFE4 (65508). Auch hier dient die Speicherstelle 780 als Register (Akku).
Zeile 10: Die Systemroutine wird aktiviert, der Computer erwartet einen Tastendruck.
Zeile 13: Unerlaubte Tasten (s. get.bas, Zeile 20) haben keine Chance: CHR$-Code 17 ist <CRSR abwärts>, Nr. 145 entspricht der Taste <CRSR aufwärts> und 19 bezeichnet die HOME-Taste. Befinden sich diese Werte im Akkumulator, wird der Inhalt von Adresse 780 gelöscht (POKE 780,0).
Zeile 18: Falls der Akkumulator (780) leer ist, kehrt das Programm zur Eingaberoutine zurück.
Zeile 20: Nach Druck auf die Taste <RETURN> verzweigt **get.sys** zu Zeile 50.
Zeile 22: Gib den Inhalt der Speicherstelle 780 als CHR-Code aus!
Zeile 25: Bilde eine Zeichenkette aus den Werten der jeweils gedrückten Tasten!
Zeile 40 bis 50: stimmen mit der Erläuterung zu **get.bas** überein.

Kurzinfo: not

Man kann nach der Eingabe eines Zahlenwerts die Zustände der Variablen Z auf dem Bildschirm betrachten.

Zeile 10: Eine INPUT-Abfrage fordert Sie auf, eine Zahl Z zwischen 1 und 5 einzugeben.

Zeile 20: Aus den drei möglichen Zuständen von Z bildet man die Summe S.

Zeile 30: Die genannten Zustände sollen auf dem Bildschirm ausgegeben werden.

Zeile 40: Besitzt die Summe S den Wert -2, haben Sie eine unerlaubt kleine Zahl eingegeben,

Zeile 50: ist die S -1, war die Zahl zu hoch!

Zeile 60: Erst, wenn S -3 war, ist der Computer zufrieden und meldet O.K.!

men, damit bei erfüllter Bedingung bestimmte Programmodule aufgerufen werden: Speziell Auswahlmenüs bieten durchnumerierte Menüpunkte, die man mit den Zahlentasten aktivieren kann. Hier wäre es unnötige Speicherplatzverschwendung, jeder Taste eine separate IF-THEN-Abfrage zu widmen, etwa wie in diesem (miserablen) Beispiel:

```
IF T$="1" THEN GOSUB 1000
IF T$="2" THEN GOSUB 2000
IF T$="3" THEN GOSUB 3000.
..
```

Basic 2.0 stellt hier zwei komfortable Befehle zur Verfügung:

ON...GOSUB/ON...GOTO-Anweisung

Voraussetzung: Dem Befehlswort »ON« muß immer eine **numerische** Variable folgen, die bei »1« zu zählen beginnt und bei maximal »9« endet, wenn Sie nur nach **einem** Tastendruck fragen. Normalerweise aber sind neun Menüpunkte für die meisten Anwendungen ausreichend. Ausgerüstet mit dem Wissen um die GET-Routine kann man problemlos mehrere Tastenbewegungen abfragen und das Programm reagieren lassen. Die Abfrage für unser IF-THEN-Beispiel sieht jetzt völlig anders aus:

```
ON X GOSUB 1000,2000,3000.
.."
```

Wird die Zahlentaste als Buchstabe von A bis Z eingelesen (z.B. T$), muß man diesen in der ON...GOSUB-Anweisung in eine Zahl umwandeln:

```
ON ASC(T$)-64 GOSUB 1000,
2000...
```

Die numerische Variable hinter ON darf auch das Ergebnis einer Berechnung sein, allerdings berücksichtigt der Basic-Interpreter dabei nur Integerzahlen: Nachkommastellen werden abgeschnitten!

Die beiden Demo-Programme **on.ziffer** und **on.letter** zeigen Beispiele, wie man Menüs mit Ziffern- und Buchstabentasten aktiviert. Basic kann also Sprünge auf beliebige Zeilen innerhalb des Programms ausführen. Die Voraussetzung zu diesem Hüpfen muß keineswegs immer solch strengen Regeln unterworfen sein, die ON-GOSUB/GOTO fordert: eine aufsteigende Reihenfolge, die bei »1« beginnt.

GOTO-Befehl

Sie können diese Anweisung auch getrennt schreiben (GO TO). Dahinter muß eine Zeilennummer stehen, zu der der C64 springt und dort mit dem Programm fortfährt. Der Befehl kann auch im Direktmodus angewendet werden. Die Zeilennummer darf getrost niedriger sein als die, in der die GOTO-Anweisung steht. Durch Rücksprünge lassen sich sogar Programmschleifen (ohne FOR-NEXT) erzeugen. Findet der Basic-Interpreter die hinter GOTO stehende Zeilennummer nicht, bricht er mit dieser Fehlermeldung ab: Undef'd Statement Error.

Im Vergleich zu noch komfortableren Basic-Dialekten fehlt Basic 2.0 die Option, GOTO auf einen variablen Wert weisen zu lassen, z.B. GOTO X. Der Befehl verlangt unbedingt eine Ganzzahl als Hinweis auf die Zeilennummer. Folgendes Programm ist nicht möglich:

```
10 x=100
20 goto x
100 print"test": end
```

Dazu finden Sie eine Mini-Basic-Erweiterung (sie funktioniert allerdings nur in Maschinensprache) auf unserer Diskette: **goto.x**. Laden Sie das Programm, und starten Sie es mit RUN. Ab sofort können Sie die Ganzzahl hinter der GOTO-Anweisung durch eine berechenbare Variable ersetzen. Als Maschinenroutine in eigenen Basic-Programmen sollte sie gleich am Anfang stehen oder als Unterprogramm definiert sein. Sie kennt außerdem noch die Anweisungen GOSUB X und RESTORE X. Achtung: Die Anweisung NEW in Zeile 24 müssen Sie unbedingt löschen!

GOSUB-RETURN-Befehl

Das Stichwort ist bereits gefallen: Unterprogramm. Selbstverständlich läßt sich so ein Programmteil auch mit GOTO aufrufen. Das hat aber einen Nachteil: Um wieder in die aufrufenden Programmzeilen zurückzukehren, muß im Unterprogramm ein weiterer GOTO-Befehl verankert sein, der den Rücksprung veranlaßt. Ein Beispiel soll das verdeutlichen:

```
10 printchr$(147)
20 print"Das ist Text 1"
30 goto 100
40 print"Das ist Text 2"
50 goto 100
60 print"Das ist Text 3"
70 goto 100
100 print:print"Taste"
110 poke 198,0:wait 198,1
120 printchr$(147)
130 goto 40
```

Kurzinfo: on.ziffer/on.letters

Zeile 10: Der Bildschirm wird gelöscht und auf Kleinschrift umgeschaltet. Die Variable FL erhält den Wert 0. Eine Erklärung, warum wir diese Variable unbedingt brauchen, folgt nach der Programmbeschreibung!

Zeile 20 bis 50: Eine Schleife von 1 bis 9 gibt diese Zahlen mit der lapidaren Bemerkung »Menüpunkt« aus.

Zeile 60: Von der GET-Routine wird jetzt Ihr Tastendruck erwartet und in der Variablen X abgelegt.

Zeile 70: Je nach Tastenwert (1 bis 9) springt das Programm zum gewünschten Menüpunkt. Diese Unterprogramme wurden zur besseren Übersicht in den Zeilen 1000 bis 9000 untergebracht.

Zeile 80 bis 85: Die Variable FL bekommt jetzt Arbeit. Besitzt sie den Anfangswert 0, springt das Programm zu Zeile 60. Hat sie den Inhalt 1 (das geschieht immer, wenn nach Tastendruck ein Unterprogramm aufgerufen wurde), verzweigt der Computer zum Programmbeginn (Zeile 10).

Zeile 100 bis 120: Dieses Unterprogramm ruft der Computer nach Aktivierung eines der neun Menüpunkte auf und gibt den Hinweis aus, daß man nach <SPACE> zum Menü zurückkehren kann. Beachten Sie: In Zeile 120 wird FL mit dem Wert 1 ausgestattet!

Zeilen 1000 bis 9000: Dies sind lediglich Beispielsmenüpunkte. Sie aktivieren das Unterprogramm in Zeile 100 und kehren mit RETURN zum Menü zurück.

Nun zur Erläuterung der ominösen Variablen FL: Sie dient als Flag (Flagge) für die Zeilen 80 und 85. Normalerweise würde genügen, wenn in Zeile 80 die Anweisung »GOTO 60« stehen würde (verzweige zur GET-Abfrage). Da aber in Zeile 70 ON mit GOSUB aufgerufen wurde, müssen die Unterprogramme in den Zeilen 1000 bis 9000 immer mit RETURN verlassen werden. Das hätte zur Folge, daß der Computer erneut in die Zeilen 60 bis 80 zurückkehren würde, jedoch niemals wieder das Menü ab Zeile auf den Bildschirm brächte! Sie könnten das umgehen, wenn Sie jeder Unterprogrammzeile (1000 bis 9000) statt mit RETURN mit »GOTO 10« abschließen würden. Doch in diesem Fall wird die GOSUB-Anweisung nicht sauber verlassen, das geht nur über RETURN: Damit zieht der Computer die Rücksprungadresse wieder von seinem Stapelspeicher (Stack) ab. Wenn Sie die Unterprogramme mit GOTO gewaltsam verlassen, wird nach einigen Menüpunktaufrufen der Stapel überlaufen (Overflow Error)! Auf solche wichtigen Kleinigkeiten sollten Sie bei eigenen Programmentwicklungen achten!

Das Programmbeispiel **on.letters** macht nichts anderes als das beschriebene, nur sind die Menüpunkte jetzt mit den Buchstabentasten A bis I aufzurufen.

Das Programm soll den Bildschirm löschen (Zeile 10) und drei Texte ausgeben (Zeilen 20, 40 und 60). Dazwischen muß es auf einen Tastendruck warten. Der Bildschirm wird erneut gelöscht, und der nächste Text erscheint auf dem Bildschirm. Um Platz zu sparen, wurde der ständig wiederkehrende Aufruf der Taste und das Bildschirmlöschen in einem Unterprogramm ab Zeile 100 untergebracht: Zeile 110 überprüft, ob eine Taste gedrückt ist, und Zeile 120 löscht den Bildschirm. In dieser Form (mit GOTO) wird das Programm nie korrekt funktionieren, weil die Rücksprunganweisung (Zeile 130) stets auf die Ausgabe von Text 2 weist. Alle nachfolgenden Programmteile werden niemals aktiviert! Das ist der Kernpunkt: Unterprogramme müssen sich aus verschiedenen Positionen des Hauptprogramms aufrufen lassen und nach getaner Arbeit wieder zum aufrufenden Programmteil zurückkehren. Eine Möglichkeit ist die Verwendung der Basic-Erweiterung **goto.x**. Dazu muß man diese zusätzlichen Programmzeilen einfügen:

```
25 x=50
55 x=80
85 x=95
95 end
```

Außerdem dürfen Sie nicht vergessen, in Zeile 130 den Wert »40« durch die Variable X zu ersetzen. Jetzt funktioniert das Programm wie gewünscht. Aber warum so umständlich? Ändern Sie das Listing erneut:

```
30 gosub 100
60 gosub 100
90 gosub 100
130 return
```

Löschen Sie noch die Zeilen 25, 55 und 85 (sie waren nur für goto.x wichtig!). Nach RUN wird der Vorteil von GOSUB gegenüber GOTO sichtbar deutlich! Wer dieses kurze Programmbeispiel nicht abtippen möchte, findet es unter dem Namen **gosub** auf unserer Diskette.

Innere und äußere Schleifen

Was haben z.B. Abfragen mit der Programmierung von Basic-Schleifen zu tun? Eine ganze Menge: Man kann mit Schleifen viel Speicherplatz sparen und Programme tunen. Basic 2.0 kennt dazu Anweisungen, die unmittelbar zusammenhängen:

FOR-TO-STEP-NEXT-Befehl

Nirgends sonst wird in Basic-Programmen die Variable »i« so oft verwendet: Sie dient als Laufvariable (Schleifenzähler). Selbstverständlich ist jeder andere numerische Variablenausdruck möglich! Wenn die Laufvariable den Wert erreicht hat, der hinter TO steht und auf die Anweisung NEXT trifft, bricht die Schleife ab: Die Aufgabe ist erfüllt. Das Programm macht mit dem nächsten Basic-Befehl weiter. Vor allem bei der Dateiverwaltung (Erfassung von Eingaben mit INPUT) oder beim Einlesen von Data-Zeilen kommt kein vernünftiges Basic-Programm ohne Schleifen aus. Natürlich kann man Basic-Schleifen auch ohne FOR-NEXT programmieren:

```
10 z=0
20 input"eingabe";eg$(z)
30 z=z+1
35 if z=10 then end
40 goto 20
```

In Zeile 10 wird die Laufvariable auf »0« gesetzt. Zeile 20 bringt den INPUT-Befehl und weist die Eingabe der eindimensionalen Variablen EG$(Z) zu. In Zeile 30 erhöht sich der Wert von Z. Um dem Computer mitzuteilen, wann er mit dem Zählen aufhören soll, steht in Zeile 35 der Endwert 10 innerhalb einer IF-THEN-Anweisung. Zeile 40 wird solange ausgeführt, bis die in Zeile 35 geforderte Bedingung wahr ist: Z = 10. Folgende Basic-Zeilen mit einer FOR-NEXT-Schleife erfüllen denselben Zweck:

```
10 for z=1 to 10
20 input"eingabe";eg$(z)
30 next
```

Diesmal sind wir mit nur drei Basic-Zeilen ausgekommen. Diese Routine ließe sich problemlos auch in einer Zeile zusammenfassen. Alle Anweisungen innerhalb einer Schleife, die nach FOR-TO stehen, werden bis zur NEXT-Anweisung ausgeführt. Dann erhöht das Programm die Laufvariable um die angegebene Schrittweite STEP und durchläuft erneut die Schleifenroutine. Fehlt STEP, erhöht der Computer den Schleifenzähler automatisch um »1«.

Schleifen kann man verschachteln. Dabei gilt: Die Schleife, die als erste geöffnet wurde, muß als letzte geschlossen werden! Unser Beispiel enthält einen Kardinalfehler:

```
10 for a=1 to 10
20 for b=1 to 5
30 print a,b
40 next a
50 next b
```

Nach RUN erhalten Sie eine Fehlermeldung: Next without for. Das Programm öffnet die Schleifen A und B, arbeitet A zehnmal ab (das bewirkt das NEXT in Zeile 40), findet aber kein NEXT zu Schleife B (diese Anweisung steht nämlich in Zeile 50!). Ist Schleife A korrekt bearbeitet, hat das Programm die FOR-Anweisung von Schleife B bereits vergessen und betrachtet das NEXT in Zeile 50 als eigenständigen Befehl (ohne FOR). Vertauschen Sie die Inhalte der Zeile 40 und 50, dann klappt's! Als Beispiel für eine verschachtelte Schleife finden Sie auf der Diskette das Programm **schachtel**: Es erzeugt ein Liniengitter auf dem Bildschirm (**Abb. 4**).

Eine FOR-NEXT-Schleife besitzt immer diese Schreibweise:

```
FOR Schleifenvariable =
Anfangswert TO Endwert
STEP Schrittweite
.
(Befehle innerhalb der
Schleife)
.
NEXT Schleifenvariable
```

Die Schrittweite läßt sich beliebig einstellen:

```
FOR T=0 TO 14 STEP 2
```

setzt die Laufvariable auf Null und zählt bis 14, aber in Zweierschritten: Die Befehle in der Schleife werden also nur siebenmal (14:2) ausgeführt!

Auch rückwärts geht's:

```
FOR T=14 TO 0 STEP -1
```

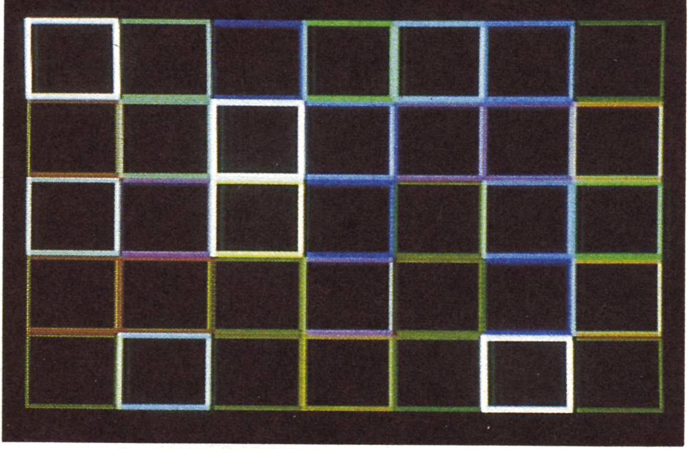
Hier muß die gewünschte Schrittweite als Minuszahl angegeben werden.

Dezimalbrüche sind ebenfalls möglich:

```
10 for t=0 to 20 step .66
20 print t
30 next
```

Beachten Sie: Der NEXT-Befehl prüft nie, ob die Laufvariable bereits den eingestellten Wert angenommen hat, sondern nur, ob sie größer ist! Erst dann verläßt das Programm die Schleife. Wenn Sie das Basic-Programm mit RUN gestartet haben und alle 0,66-Werte zwischen »1« und »20« auf dem Bildschirm erschienen sind, überprüfen Sie den Schleifenzähler:

```
PRINT T
```



[4] Verschachtelte Schleifen erzeugen farbige Rechtecke

Als Ergebnis erhalten Sie: 20.14 (statt 20). Bei manchen Programmabläufen kann es jetzt nötig werden, die Variable T wieder auf den gewünschten Wert zurückzusetzen (z.B. T = 20)!

Wie spät ist es, bitte?

Der C64 besitzt eine innere Uhr, die man abfragen, zu Messungen oder zur Programmsteuerung verwenden kann. Sie startet nach dem Einschalten immer mit »0«. Diese beiden Systemvariablen sind zuständig:

TI und TI$

Die interne Uhr des C64 läßt sich beeinflussen: Will man den aktuellen Stand wissen, fragt man den Inhalt der Variablen TI ab. Dazu gibt's eine Endlosschleife, die jeder Basic-Einsteiger kennt:

```
10 print ti: goto 10
```

Auf dem Bildschirm sieht man nach dem Start mit RUN ein »ewiges« Zahlenband mit sich ständig ändernden Werten.

Diese Zahl, geteilt durch 60, gibt uns die Zeit an, die seit dem Einschalten des Computers oder Umstellen der internen Uhr vergangen ist. Will man die Minuten wissen, muß man die Zahl von TI durch 3600 teilen, bei Stunden durch 21600. Wer schwach im Kopfrechnen ist, tut sich mit der anderen Systemvariablen des Basic-Interpreter leichter: TI$. Sie gibt die Zeit ebenfalls sechsstellig aus, aber bereits umgerechnet in Stunden, Minuten und Sekunden. Der Wert »124533« bedeutet beispielsweise: 12 h, 45 min und 33 s. Mit dieser Anweisung erscheint TI$ auf dem Bildschirm:

```
10 printti$: goto 10
```

Das Zahlenband, das jetzt durchläuft, verändert sich nur noch im Sekundenrhythmus. Durch den Einsatz von TI$ läßt sich die innere Uhr auch stellen:

```
TI$="000000"
```

setzt sie auf den Anfangswert »0«,

```
TI$="233000"
```

beispielsweise auf 23 Uhr 30.

Kurzinfo: stoppuhr

Zeile 110 setzt die Zeitvariable auf 0, dann wird der Bildschirm gelöscht. In **Zeile 140** schreibt der Computer hintereinander jede Menge Sternchen auf den Bildschirm. **Zeile 150** prüft, ob eine Taste gedrückt ist oder nicht (wenn nicht, hat die dafür zuständige Speicherstelle 203 stets den Inhalt »64«). In **Zeile 160** springt das Programm zurück zu 140. Wurde eine beliebige Taste gedrückt, hüpft das Programm zu **Zeile 180**, gibt einen Zeilenvorschub auf dem Bildschirm und anschließend die Sekundenzahl für die Zeitspanne aus, die seit dem Start mit RUN vergangen ist. Wichtig: Wenn TI den Wert »5184000« annimmt, entspricht dies einer Laufzeit von 24 h. Wie bei einer echten Uhr bekommt die Systemvariable jetzt wieder den Wert 0: Ein neuer Tag beginnt... Das läßt sich mit folgendem Listing leicht überprüfen:

```
10 ti$="235930"
20 printti$,ti
30 goto 20
```

Wenn Sie das Programm nach dem Start mit RUN 30 s laufen lassen, können Sie auf dem Bildschirm beobachten, wie die Zeit kippt.

Nach <RETURN> beginnt die Uhr mit der Zeitnahme. Folgendes Listing (es befindet sich auf der Diskette) simuliert eine Stoppuhr:

```
110 ti$="000000"
120 printchr$(147)
140 print "*";
150 ifpeek(203)<>64
then180
160 goto 140
180 print
190 print ti/60"sek"
200 end
```

Die Erläuterung finden Sie im entsprechenden Kurzinfo-Kasten. Eine andere Form der Zeitübersicht am Computer bringt ein weiteres Basic-Programm auf der Diskette: **zeitanzeige**.

Rein zufällig...

Nicht um fixierte, sondern um zufällige Werte geht's in unserem nächsten Kapitel:

RND-Funktion

Sie nimmt sofort nach dem Einschalten des Computers ihre Arbeit auf. Wenn Sie die Funktion nicht aktivieren, bleibt sie natürlich unsichtbar und wirkt nur im Hintergrund. Damit generiert der Computer eine Zufallszahl zwischen 0 und 1. Gerade bei Spielen und Grafik, seltener bei mathematischen Berechnungen, ist es oft nötig, von zufällig entstandenen Zahlen auszugehen. Die Basic-Funktion lautet: RND(X). Das X in Klammern ist das Argument. Es kann drei unterschiedliche Werte besitzen:

- positiv (egal, welche Zahl),
- negativ,
- die Zahl 0.

Experimentieren Sie mit folgenden Programmzeilen:

```
10 a=rnd(1)
20 print a
30 goto 10
```

Kurzinfo: rnd(0)/rnd(1)

Gesteuert durch diese Zufallsfunktion erscheinen farbige Blöcke auf dem Bildschirm.

Zeile 10: stellt die Farbe Schwarz für Hintergrund und Rahmen ein, löscht den Bildschirm.

Zeile 30: In einer Schleife wird der Bildschirm mit dem Wert »160« (=inverser Block, <SHIFT SPACE>) beschrieben, der ebenfalls schwarz ist.

Zeile 40: Ins Farb-RAM ab 55296 POKEt das Programm nun zufällige Farbwerte. Sie ändern die Farben der <SHIFT SPACE>-Zeichen auf dem Bildschirm, ebenfalls an zufällig gewählten Positionen.

Zeile 45: Falls Sie eine Taste drücken (Inhalt von 203 anders als 64), endet das Programm.

Zeile 50: Endlosschleife zu Zeile 40.

Nach einer gewissen Zeit stellt man fest, daß sich trotz rein zufälligen RND-Anweisungen eine sichtbare Ordnung auf dem Bildschirm entwickelt (Abb.)

Beim Listing **rnd(1)** liegt der einzige Programmunterschied im Argument des RND-Befehls. Hier läßt sich aber deutlich erkennen, daß die Farbwerte echt zufällig auf dem Bildschirm verteilt werden (Abb. 5 u. 6).

Kurzinfo: zeitanzeige

Zeile 20 bis 30: fordert Sie auf, die aktuelle Zeit im Format HH.MM.SS (Stunden, Minuten und Sekunden mit zwei Ziffern, jeweils durch einen Punkt getrennt), einzugeben.

Zeile 30: Das ist die INPUT-Routine, ausgestattet mit einem kleinen Trick: POKE 19,64 vor dem INPUT-Befehl unterdrückt das lästige Fragezeichen. Ihre Eingabe wird in der Variablen ZT$ gespeichert.

Zeile 50: bearbeitet die Eingabe und weist sie TI$ zu. Da die eintippten Punkte zwischen den Zeiteinheiten bei TI$ einen »Syntax Error« erzeugen würden, muß Ihre Eingabe in drei Teile aufgegliedert werden: Mit MID$ (s. Beschreibung!)

Zeile 60: positioniert die Cursorposition in Spalte 32, Zeile 0 des Bildschirms.

Zeile 70: Die Variable TI$ wird erneut per MID$ in drei Teile gespalten, um dazwischen Doppelpunkte zu plazieren (wie bei Digitalanzeigen üblich!)

Zeile 80: Wurde keine Taste gedrückt (dann ist der Inhalt von Adresse 203 immer »64«), zeigt das Programm die laufende Zeit an (Sprung zu Zeile 60 und 70).

Nach dem Start mit RUN ist der C64 allerdings von diesem Basic-Programm blockiert: Sie können nichts anders mit ihm anstellen. Dazu müßte die Zeitangabe durch ein Maschinenprogramm erzeugt werden, das in den Interrupt eingebunden ist...

Wie schon seit dem Kapitel über TI und TI$ bekannt, liefert der Computer jetzt wieder eine Zahlenkolonne. Alle Werte liegen zwischen 0 und 1 (z.B. .450032866). Ab und zu tauchen in der Zahlenreihe merkwürdige Ziffernfolgen auf: mit einer Zahl vor dem Dezimalpunkt und einem »E«, dem eine Minuszahl folgt, z.B. »3.80238951E -03«. Hier handelt es sich um Gleitkommakonstanten in der Exponentialdarstellung. Sie bestehen aus einer ganzen Zahl oder Festkommazahl (Mantisse), gefolgt vom Buchstaben »E« und einer ganzen Zahl im Bereich zwischen -39 und +38. Dieser Wert ist der Exponent zur Basic 10. Verständlich

So finden Sie die Programme auf der Diskette

DISKETTE SEITE 1

```
0    [illegible]
24   "disklader"            prg      Seite 21
0    "  ┌──────────────┐"   usr
0    "  |    basic-    |"   usr
0    "  | erweiterungen|"   usr
0    "  └──────────────┘"   usr
58   "s+g-basic"            prg      Seite 38
2    "line1"                prg
2    "line2"                prg
5    "flugzeug"             prg
1    "arc"                  prg
1    "rec"                  prg
6    "spritedemo1"          prg
24   "spritedemo2"          prg
3    "grafikdemo/1"         prg
3    "grafikdemo/2"         prg
3    "grafikdemo/3"         prg
3    "grafikdemo/4"         prg
3    "grafikdemo/5"         prg
2    "grafikdemo/6"         prg
0    "----------------"     usr
20   "xbasic plus"          prg      Seite 43
1    "xbasic.demo1"         prg
0    "  ┌──────────────┐"   usr
0    " |aufloesung der |"   usr
0    " |   knobelecke  |"   usr
0    " | sonderheft 56 |"   usr
0    "  └──────────────┘"   usr
138  "beste wege"           prg      Seite 32
1    "drucker"              seq
9    "leer"                 usr
9    "bytehausen"           usr
9    "bytehausen-opti"      usr
9    "kleinstadt i"         usr
9    "kleinstadt ii"        usr
9    "kleinstadt iii"       usr
9    "schlingenhausen"      usr
0    "  ┌──────────────┐"   usr
0    " |   diskette    |"   usr
0    " |  beidseitig   |"   usr
0    " |   bespielt    |"   usr
0    "  └──────────────┘"   usr
```

DISKETTE SEITE 2

```
0    [illegible]
0    "  ┌──────────────┐"   usr
0    " |  basic-kurs   |"   usr      Seite 4
0    " | zum mitmachen |"   usr
0    "  └──────────────┘"   usr
2    "basic-befehle"        prg
2    "fehlermeldungen"      prg
1    "einschaltmeldung"     prg
2    "eindim"               prg
2    "zweidim"              prg
1    "input.sys"            prg
1    "get.bas"              prg
1    "get.sys"              prg
2    "get.curs"             prg
1    "not"                  prg
3    "on.ziffer"            prg
3    "on.letters"           prg
2    "goto x"               prg
1    "gosub"                prg
1    "stoppuhr"             prg
2    "zeitanzeige"          prg
2    "schachtel"            prg
1    "rnd(0)"               prg
1    "rnd(1)"               prg
1    "wuerfelzahlen"        prg
2    "rnd-lotto"            prg
6    "lotto"                prg
1    "read-data"            prg
1    "rnd-read"             prg
6    "begriffe raten"       prg
1    "farbwechsel"          prg
1    "rnd-farben"           prg
1    "farbrand"             prg
2    "farbumlauf"           prg
2    "zeilenkiller"         prg
1    "sys-zeit"             prg
1    "usr(0)"               prg
2    "blockmove"            prg
2    "sys-save"             prg
1    "write.seq"            prg
1    "datei"                seq
1    "read.seq"             prg
1    "print#"               prg
1    "get#"                 prg
1    "sport"                seq
6    "literatur-datei"      prg
4    "literatur"            seq
5    "setpixel"             prg
7    "draw.bas"             prg
8    "circle.bas"           prg
10   "box.bas"              prg
11   "fussballplatz"        prg
8    "circle.multi"         prg
0    "  ┌──────────────┐"   usr
0    " | tips & tricks |"   usr      Seite 35
0    "  └──────────────┘"   usr
3    "speedy hires"         prg
0    "----------------"     usr
1    "char-lader"           prg
17   "amiga.char"           prg
17   "game.char"            prg
0    "----------------"     usr
1    "bildsch.langsam"      prg
1    "bs/ascii-code"        prg
1    "88 zeichen"           prg
3    "restore.bas"          prg
1    "scroll n. unten"      prg
5    "zahlenwandler"        prg
0    "----------------"     usr
19   "rolling"              prg
3    "roll.obj"             prg
17   "zs"                   prg
0    "----------------"     usr
1    "auto.bas"             prg
1    "dir.bas"              prg
1    "nullen"               prg
1    "minidisass"           prg
0    "----------------"     usr
3    "pulldown-menue"       prg
5    "pull-down"            prg
0    "  ┌──────────────┐"   usr
0    " |     ende      |"   usr
0    "  └──────────────┘"   usr
```

WICHTIGE HINWEISE

zur beiliegenden Diskette:

Aus den Erfahrungen der bisherigen Sonderhefte mit Diskette wollen wir ein paar Tips an Sie weitergeben:

1 **Bevor Sie mit den Programmen auf der Diskette arbeiten, sollten Sie unbedingt eine Sicherheitskopie der Diskette anlegen. Verwenden Sie dazu ein beliebiges Kopierprogramm, das eine komplette Diskettenseite dupliziert.**

2 **Auf der Originaldiskette ist wegen der umfangreichen Programme nur wenig Speicherplatz frei. Dies führt bei den Anwendungen, die Daten auf die Diskette speichern, zu Speicherplatz-Problemen. Kopieren Sie daher das Programm, mit dem Sie arbeiten wollen, mit einem File-Copy-Programm auf eine leere, formatierte Diskette und nutzen Sie diese als Arbeitsdiskette.**

3 **Die Rückseite der Originaldiskette ist schreibgeschützt. Wenn Sie auf dieser Seite speichern wollen, müssen Sie vorher mit einem Diskettenlocher eine Kerbe an der linken oberen Seite der Diskette anbringen, um den Schreibschutz zu entfernen. Probleme lassen sich von vornherein vermeiden, wenn Sie die Hinweise unter Punkt 2 beachten.**

ALLE PROGRAMME
aus diesem Heft



HIER

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Redaktionsdirektor: Dr. Manfred Gindle

Chefredakteur: Georg Klinge – verantwortlich für den redaktionellen Teil
Stellv. Chefredakteur: Arnd Wängler
Textchef: Jens Maasberg
Produktion: Andrea Pfliegensdörfer
Redaktion: Harald Beiler (bl), Herbert Großer (gr)
Mitarbeiter der Ausgabe: Nikolaus Heusler
Redaktionsassistenz: Sylvia Wilhelm, Birgit Misera (089/4613202)
Telefax: 089/4613-5001
Alle Artikel sind mit dem Kurzzeichen des Redakteurs und/oder mit dem Namen des Autors/Mitarbeiters gekennzeichnet

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten worden sein, muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt&Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt&Technik Verlag AG verlegten Publikationen. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Verlagsleitung: Wolfram Höfler
Operation Manager: Michael Koeppe

Layout: Axel Waldhier
Bildredaktion: Walter Linne (Fotografie); Ewald Standke, Norbert Raab (Spritzgrafik); Werner Nienstedt (Computergrafik)

Anzeigendirektion: Jens Berendsen
Anzeigenleitung: Philipp Schiede (399) – verantwortlich für die Anzeigen
Telefax: 089/4613-775

Anzeigenverwaltung und Disposition: Chris Mark (421)

Auslandsrepräsentation:
Auslandsniederlassungen:
Schweiz: Markt&Technik Vertriebs AG, Kollerstr. 37, CH-6300 Zug, Tel. 0041/42-440550, Telefax 0041/42-415770
USA: M&T Publishing Inc.; 501 Galveston Drive Redwood City, CA 94063, Telefon: (415) 366-3600, Telefax 415-3663923
Österreich: Markt&Technik Ges. mbH, Große Neugasse 28, A 1040-Wien Telefon: 0043/1/58713930, Telefax: 0043-1-587139333

Anzeigen-Auslandsvertretung:
Großbritannien: Smyth Int. Media Representatives, Telefon 0044/81340-5058, Telefax 0044/81341-9602

Israel: Baruch Schaefer, Telefon 00972-3-5562256, Telefax 00972/52/444518
Taiwan: AIM Int. Inc., Telefon 00886-2-7548613, Telefax 00886-2-7548710
Japan: Media Sales Japan, Telefon 0081/33504-1925, Telefax 0081/33596-1709
Korea: Young Media Inc., Telefon 0082-2-7564819, Telefax 0082-2-7575789
Frankreich: CEP France, Telefon 0033/148007616, Telefax 0033/148240202
Italien: CEP Italia, Telefon 0039/24982997, Telefax 0039/24692834
International Business Manager: Stefan Grajer 089/4613-638

Gesamtvertriebsleiter: York v. Heimburg
Vertriebsmarketing: Helmut Pleyer (710)

Vertrieb Handel: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: ip Internationale Presse, Ludwigstraße 26, 7000 Stuttgart 1, Tel. 0711/619660

Einzelheft-Bestellung: Markt & Technik Leserservice, CSJ Postfach 140220, 8000 München 5

Verkaufspreis: Das Einzelheft kostet DM 16,–

Produktion: Technik: Klaus Buck (Ltg./180), Wolfgang Meyer (Stellv./887);

Druck: SOV Graphische Betriebe, Laubanger 23, 8600 Bamberg

Sonderdruck-Dienst: Alle in dieser Ausgabe erschienenen Beiträge sind in Form von Sonderdrucken zu erhalten. Anfragen an Reinhard Jarczok, Tel. 089/4613-185, Fax 4613-774.

Vorstand: Otmar Weber (Vors.), Bernd Balzer, Dr. Rainer Doll, Lutz Glandt

Direktor Zeitschriften: Michael M. Pauly

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen: Markt&Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon 089/4613-0, Telex 522052, Telefax 089/4613-100

ISSN 0931-8933

Copyright-Erklärung

Name: ....

Anschrift: ....

Datum: ....

Computertyp: ....

Benötigte Erweiterung/Peripherie: ....

Datenträger: Kassette/Diskette ....

Programmart: ....

Ich habe das 18. Lebensjahr bereits vollendet

...., den ....

....
(Unterschrift)

Wir geben diese Erklärung für unser minderjähriges Kind als dessen gesetzliche Vertreter ab.

...., den ....

....

....

Bankverbindung:

Bank/Postgiroamt: ....

Bankleitzahl: ....

Konto-Nummer: ....

Inhaber des Kontos: ....

Das Programm/die Bauanleitung: ....

das/die ich der Redaktion der Zeitschr[illegible] 64'er übersandt habe, habe ich selbst erarbeitet und nicht, auc[illegible] nicht teilweise, anderen Veröffentlichungen entnommen. Das [illegible]ogramm/die Bauanleitung ist daher frei von Rechten anderer [illegible]nd liegt zur Zeit keinem anderen Verlag zur Veröffentlichung vor. [illegible]ch bin damit einverstanden, daß die Markt & Technik Verlag AG [illegible]as Programm/die Bauanleitung in ihren Zeitschriften oder ihren herausgegebenen Büchern abdruckt und das Programm/die Bauanleitung vervielfältigt, wie beispielsweise durch Herstellung von Disketten, auf denen das Programm gespeichert ist, oder daß sie Geräte und Bauelemente nach der Bauanleitung herstellen läßt und vertreibt bzw. durch Dritte vertreiben läßt.

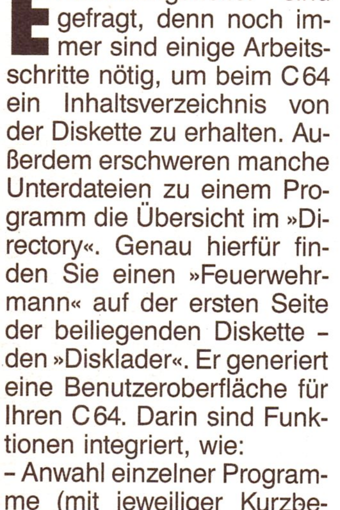
Ich erhalte, wenn die Markt & Technik Verlag AG das Programm/die Bauanleitung druckt oder sonst verwertet, ein Pauschalhonorar.

Disklader – Programme laden mit Komfort

Diskettenoberfläche de Luxe

Keine umständlichen Ladeanweisungen und ein übersichtliches Inhaltsverzeichnis der Diskette auf dem Bildschirm. Unser »Disklader« erfüllt auch gehobene Ansprüche.

von Herbert Großer



Entwicklungshelfer sind gefragt, denn noch immer sind einige Arbeitsschritte nötig, um beim C64 ein Inhaltsverzeichnis von der Diskette zu erhalten. Außerdem erschweren manche Unterdateien zu einem Programm die Übersicht im »Directory«. Genau hierfür finden Sie einen »Feuerwehrmann« auf der ersten Seite der beiliegenden Diskette – den »Disklader«. Er generiert eine Benutzeroberfläche für Ihren C64. Darin sind Funktionen integriert, wie:

- Anwahl einzelner Programme (mit jeweiliger Kurzbeschreibung),
- automatisches Laden und Starten von Diskette oder
- Erkennung der richtigen Diskette bzw. Diskettenseite.

Da sich der Disklader an erster Stelle auf der Diskette zum Sonderheft befindet, genügt es, zum Laden einzugeben:

```
LOAD":*",8
```

Nach der Bestätigung mit <RETURN> dauert es ca. 15 s, bis die Datei im Speicher ist. Sie starten mit RUN und <RETURN>. Anschließend wird das File entpackt (ca.2 s) und es erscheint die Benutzeroberfläche des »Disklader« (s. Abbildung). In der rechten unteren Bildschirmhälfte sehen Sie weiß umrandet den Namen des ausgewählten Programms. Die unterste Bildschirmzeile ist die dazugehörige Kurzerklärung. Zusätzlich finden Sie in der rechten unteren Bildschirmhälfte den Text »Seite 1 auf Disk« oder »Seite 2 auf Disk«. Da Sie die Inhaltsverzeichnisse beider Seiten (ohne die Disk zu wenden) durchblättern können, finden Sie hier

angewähltes programm – Kurzerklärung für das angewählte Programm – Seite auf der Diskette und Feld für Fehlermeldungen

den Hinweis, auf welcher Diskettenseite sich das gewählte Programm befindet.

Durch Tastendruck <CRSR aufwärts> bzw. <CRSR abwärts> wählen Sie das nächste oder vorherige Programm. Sie blättern quasi durch den Inhalt der Programme. <HOME> bringt Sie zum ersten Eintrag des Inhaltsverzeichnisses. Selbstverständlich sind nur die Programme verzeichnet, die sich eigenständig laden oder starten lassen.

<RETURN> führt Sie in den Ladeteil. Ist kein Diskettenfehler aufgetreten, erscheint kurzzeitig »00,OK, 00,00« am Bildschirm. Eventuelle Fehleranzeigen bleiben sichtbar am Bildschirm (z.B. »21,READ ERROR, 18,00«= Drive not ready). Sie lassen sich durch einen beliebigen Tastendruck wieder löschen. Schlagen Sie bitte vorher im Handbuch Ihrer Floppy nach und beseitigen Sie den Fehler. Eine andere Art der Fehlermeldung wird durch einen blinkenden Text dargestellt (z.B. »Bitte Disk wenden« oder »Falsche Diskette«). Sind Fehler ausgeblieben, lädt der Disklader das von Ihnen gewählte Programm von der Diskette und startet es. Ladefehler, die in dieser Phase auftreten, werden nicht mehr berücksichtigt: Der Disklader wird vom neuen Programm einfach überschrieben. Sonst könnten wir nur Programme veröffentlichen, die mit der Benutzeroberfläche zusammenarbeiten. Bei vielen Spielen, Tricks oder Tools ist dies aber nicht der Fall.

Für Sie bedeutet dies, nach jedem Starten eines Programms den »Disklader« erneut zu laden. Wer die Benutzeroberfläche verlassen will, gibt <RUN/STOP> ein. Sie befinden sich dann im normalen »Basic« des C64. Für einen Neustart befehlen Sie

```
SYS 12032
```

und bestätigen mit <RETURN>. Dieser Neustart funktioniert auch nach einem Reset, d.h. wenn Sie durch den entsprechenden Taster einen Hardware-Reset ausgelöst haben. Allerdings sollten Sie zwischenzeitlich kein Programm geladen haben, da dies den verwendeten Speicherbereich überschreiben könnte. Laden Sie in diesem Falle den Disklader neu.

Wir haben bei der Programmierung größten Wert auf Kompatibilität mit den unterschiedlichsten Betriebssystemerweiterungen gelegt. Lediglich bei der Gerätekonfiguration C128 mit RAM-Erweiterung und zweiter Diskettenstation sollten Sie die externe Floppy ausschalten. (gr)

Kurzinfo: Disklader

Programmart: Hilfsprogramm zum Laden der Programme auf der beiliegenden Diskette
Laden: LOAD":*",8
Starten: nach dem Laden mit RUN
Steuerung: Tastatur
Programmautor: H. Großer

ausgedrückt bedeutet die Zahl »3.80238951E-03« dasselbe wie »0.0038023895«.

Ein positives Argument X beginnt immer mit demselben Anfangswert. Ist X=0, nimmt der Computer als Anfangszahl den aktuellen Stand der inneren Computeruhr. Bei negativen Argumenten bildet diese Zahl selbst den Startwert. Wir empfehlen: Verwenden Sie für RND stets 1! RND(0) kann nämlich nur 256 verschiedene Zufallswerte erzeugen! Deutlich ist dieser Unterschied mit den Beispielen **rnd(0)** und **rnd(1)** (**Abb. 5 und 6**) auf dem Bildschirm zu sehen.

Ändern Sie das Argument der RND-Funktion in Zeile 10 unseres Programmbeispiels auch in andere beliebige positive und negative Werte und vergleichen Sie die Zahlen, die auf dem Bildschirm auftauchen.

Kommastellen abgetrennt

Bisher wurden die Zufallszahlen als Brüche (mit Nachkommastellen) angezeigt. Nehmen wir an, in einem Spielprogramm (Würfeln) sollen Zahlen zwischen 1 und 6 erzeugt werden. Das Würfelergebnis der RND-Funktion (z.B. 3.57814324 oder 4.11720395) würde die Spiellaune bald verderben... Auch für diesen Fall kennt Basic 2.0 ein Rezept:

INT-Funktion

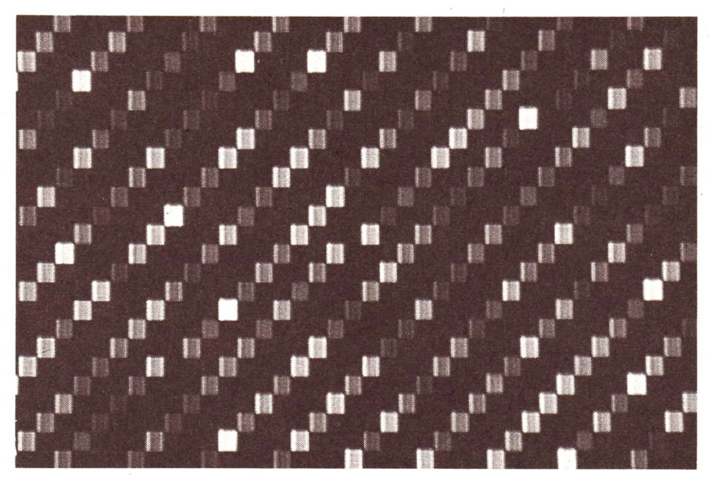
Es ist die Abkürzung von INTEGER. Daraus erkennen Sie, daß es sich um ganze Zahlen handeln muß. Schreiben Sie den Dezimalbruch (oder dessen Variablennamen) in Klammern hinter die Anweisung:

```
print int(4.1172)
```

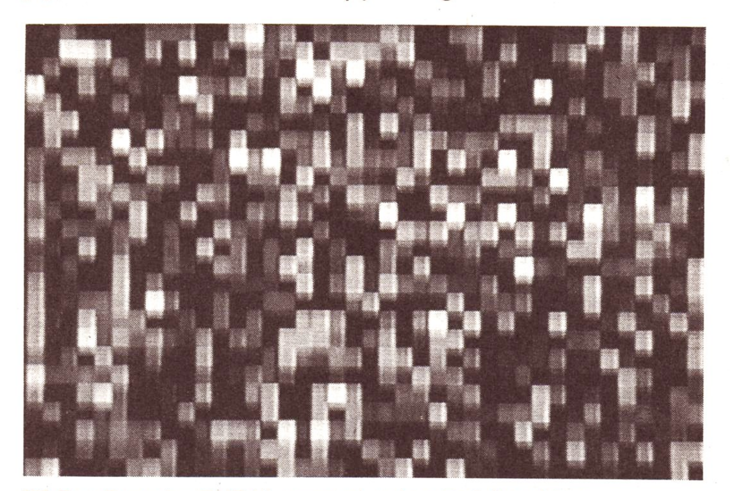
Als Ergebnis erhalten Sie »4«. Da der Wert der Nachkommastellen relativ niedrig ist, ist man mit diesem Ergebnis einverstanden, aber:

```
print int(4.99999)
```

ergibt ebenfalls »4« (obwohl die Zahl doch fast den Wert »5« besitzt!). Diese großzügige Berechnung des C 64 hat



[5] Zufall? Die Funktion RND(0) sieht geordnet aus



[6] Der Beweis: RND(1) erzeugt echte Zufallszahlen

einen simplen Grund: Er macht sich nicht die Mühe, auf- oder abzurunden – er schneidet die Kommastellen hinter der ganzen Zahl einfach ab!

Die Kommazahl, die mit INT bearbeitet werden soll, darf auch eine mathematische Formel oder ein anderer, komplizierter Ausdruck sein. Voraussetzung: Er muß sich berechnen lassen. Zwingend erforderlich sind die Klammern, in denen die Zahl oder der Ausdruck stehen muß. Wer sie wegläßt, wird mit einem »Syntax Error« bestraft. In der Praxis mit den Zufallszahlen sieht es nun so aus:

```
10 a=rnd(0)
20 print int(a)
30 goto 10
```

Das Ergebnis — blamabel! Eine Perlenkette von Nullen tummelt sich auf dem Bildschirm. Die Erklärung ist schnell gefunden: Alle RND(0)-Zahlen sind kleiner als »1«, also »0« und einige »Zerquetschte«. Sind die Nachkommastellen eliminiert, bleibt eben nur »0« übrig! Dagegen sollten Sie sofort etwas unternehmen: Zeile 20 muß so verbessert werden, daß der Dezimalpunkt nach rechts verschoben wird (durch eine Multiplikation von 10, 100, 1000 usw.):

```
20 print int(a*10)
```

erzeugt Zufallszahlen zwischen 0 und 9,

```
20 print int(a*100)
```

liefert Werte zwischen 0 und 99. Die letztgenannte Version sähe als Einzeiler so aus (Verschachtelung der Klammern und ihre exakte Position beachten!):

```
10 print int(rnd(0)*100):
   goto 10
```

Fehlt eine Klammer, erscheint unerbittlich wieder ein »Syntax Error«. Eines stört allerdings noch an dieser Programmierung der RND-Funktion: Es können auch Zahlen erscheinen, die geringer als »1« sind: Nullwerte. Unser Beispiel mit dem Würfelspiel würde damit nicht hinhauen: Ein Würfel kennt schließlich nur Zahlen von 1 bis 6! Die Lösung: Der INT-Ausdruck muß noch mit dem Wert addiert werden, der als unterste Grenze bei der Ausgabe der Zufallszahlen gelten soll:

```
5 rem würfelzahlen
10 for w=1 to 6
20 a=int(rnd(0)*6)+1
30 print a
40 next
```

Das Beispielprogramm **wuerfelzahlen** befindet sich auf der Diskette. Nach dem Start mit RUN gibt der Computer sechs Zahlen aus, die Sie quasi gewürfelt haben. Stören Sie sich nicht daran, daß ab und zu gleiche Zahlen auftauchen: Das kommt beim echten Würfeln oft genug vor. In Wirklichkeit sind die Zahlen aber nicht gleich, denn im abgeschnittenen Nachkommateil würden wir erhebliche Unterschiede entdecken!

Eine sehr beliebte Anwendung der RND-Funktion ist das Erzeugen von Lottozahlen. Warum sich jede Woche den Kopf über neue Tippzahlen zerbrechen, wenn man einen C 64 hat? Nur: Hier wirkt sich das mehrmalige Auftauchen ein und derselben Zahl unerfreulich aus: Beim Lotto

Kurzinfo: würfelzahlen

Das Programm gibt sechs Zahlen aus, die das Ergebnis von sechsmaligem Würfeln simulieren sollen.

Zeile 10: öffnet die Schleife W für sechs Durchgänge.

Zeile 20: belegt die Variable A mit einem zufälligen Wert, der nicht größer als 6 und nicht kleiner als 1 sein darf.

Zeile 30: Der jeweiligen Werte von A erscheinen sechsmal untereinander auf dem Bildschirm.

Möchten Sie nur jeweils eine gewürfelte Zahl erzeugen, genügt dieser Einzeiler:

```
10 a=int(rnd(1)*6)+1:print a
```

Kurzinfo: rnd-lotto

Nach dem Start mit RUN gibt der Computer untereinander eine Lottozahlenreihe aus, die Sie z.B. für Ihren nächsten Tippschein verwenden können. Mit der Taste <N> läßt sich eine neue Zahlenkolonne aktivieren, <E> beendet das Programm.

Zeilen 10 bis 20: Bildschirmausgabe der Überschrift »Mini-Lotto«.

Zeile 30: Die Schleife I wird geöffnet, sechs Durchgänge sind vorgesehen.

Zeile 31: Sprung zum Unterprogramm in **Zeile 90:** Dort zieht der Computer die sechs Zahlen zunächst intern und speichert sie in den indizierten Variablen ZA(1) bis ZA(6).

Zeile 33: Eine weitere Schleife (J) mit sechs Durchläufen wird vorbereitet. Grund: Die gezogenen Zahlen sollen nun in umgekehrter Reihenfolge (Schleife I, FOR I=6 TO 1 STEP-1) verglichen werden.

Zeile 34: Stellt der Computer fest, daß zwei gezogene Zahlen identisch sind, aktiviert das Programm erneut das Unterprogramm in Zeile 90. Damit erhält man nun eine andere Zahl.

Zeile 35: Zwischen den Schleifenabläufen gibt der Computer die Zahlen auf dem Bildschirm aus.

Zeilen 40 bis 80: ...fragen ab, ob erneut gezogen oder das Programm beendet werden soll.

(Vollsystem) gibt's zwar auch sechs Zahlen zwischen 1 und 49, doch müssen alle verschieden sein! Eine praktikable Lösung bieten unsere Programmbeispiele **rnd-lotto** und, in der komfortableren Version, **lotto (Abb. 7)**, aus dem 1984 erschienenen Buch »Das große Spielebuch für den C64« vom Markt & Technik-Verlag.

[7] Lottozahlen per Computer: Ist Glück berechenbar?

Datenbanken

INPUT und GET sind komfortable Anweisungen, um dem Computer Zahlen, Wörter oder Text für ein Programm zu übermitteln. Auf Dauer wird das aber mühsam, vor allem, wenn die Daten bereits feststehen, die man verarbeiten möchte. Einmal eingegeben, sollten sie immer zur Verfügung stehen.

Seit dem Kapitel über Variablen wissen wir, daß sich z.B. Strings unter einem markanten Namen im Computer speichern lassen. Um etwa fünf Zeichenketten zu definieren, muß man fünf Zuordnungen aufstellen:

```
10 a$="computer"
20 b$="monitor"
30 c$="floppy"
40 d$="joystick"
50 e$="drucker"
```

Bei fünf Strings kein Problem - aber bei 30 oder 200? Basic 2.0 kennt dazu ein komfortables Befehlswort:

DATA-Anweisung

In einer Befehlszeile lassen sich Zahlenwerte und Strings erfassen. Alle Einträge müssen durch ein Komma getrennt sein. Unsere Listingzeilen sehen nun so aus:

```
100 data computer,monitor,
floppy,joystick,drucker
```

Vorteil: Die Strings müssen nicht in Anführungszeichen stehen. Ausnahme: wenn Sie Grafikzeichen, Strings mit Kommas oder bei eingestellter Kleinschrift Großbuchstaben verwenden. Beispiel:

```
100 data "Computer",
"Monitor","eine Floppy,
die schnell ist!",
"Joystick","Drucker"
```

Mit Zahlen sieht eine DATA-Zeile so aus:

```
200 data 25,123,225,16,24
```

Die Daten, die man ständig benötigt, sind nun in Programmzeilen erfaßt und stehen sofort nach dem Start zur Verfügung. Erneute Eingabe entfällt.

Wie kommt aber der Computer an diese Daten? Mit der

READ-Anweisung

Die Übersetzung ist treffend: Read heißt nun mal Lesen. Integrieren wir diesen Befehl in unser Beispiel:

```
40 read a$
50 print a$
```

Nach RUN bringt das Programm das erste Wort: Computer. Das ist's aber auch schon - die weiteren Strings lassen sich damit nicht aktivieren. Wer z.B. in den Zeilen 40 und 50 die Variable durch eine numerische ersetzt (A), hat ebenfalls keinen Erfolg und wird zusätzlich mit einer Fehlermeldung bestraft: Syntax Error.

Wir haben fünf Stringwörter: Der READ-Befehl muß also fünfmal ausgeführt werden. Das funktioniert mit einer Schleife. Ergänzen Sie unser Programmbeispiel um folgende Zeilen:

```
30 for i=1 to 5
60 next
```

Jetzt tauchen alle fünf Strings auf. Eine Erhöhung der Schleifenvariablen I auf »10« gibt auch noch die fünf Zahlen in Zeile 200 aus. Wollen Sie mehr Strings einlesen, als in den DATA-Zeilen vorhanden sind (Schleife I wird z.B. auf »12« erhöht), erhält man die Fehlermeldung: Out of data. Verringern läßt sich der Schleifenwert jedoch jederzeit.

Der Computer hat die Zahlen in Zeile 200 zwar akzeptiert, aber für ihn sind es keine Zahlen, sondern Zeichenketten. Zu Berechnungen kann man sie in dieser Form nicht einsetzen. Dazu muß man eine weitere Schleife für numerische Variable öffnen:

```
62 for i=1 to 5
63 read a: print a
64 next
```

Der Unterschied wird auf dem Bildschirm sichtbar: Alle Zahlen sind bei der Ausgabe jetzt vom Rand um eine Stelle eingerückt (das Byte fürs Vorzeichen!)

Ein weiteres Experiment zur READ-Anweisung: Versuchen Sie, die Ausgabe der DATA-Werte endlos laufen zu lassen. Fügen Sie die nächste Zeile in unser Programmbeispiel ein:

```
70 goto 30
```

Nach RUN erscheinen zwar die Daten, aber auch eine Fehlermeldung: Out of data. Schuld dran ist die Verfahrensweise des READ-Befehls: Er läßt einen internen Zähler mitlaufen, der ständig anzeigt, auf welchen DATA-Wert das Programm jetzt zugreifen soll. Nachdem beide Schleifen abgearbeitet sind, bleibt der Zähler bei seinem erhöhten Wert und versucht, den Wert Nr. 11 zu holen (den gibt's aber nicht!). Folgender Basic-Befehl behebt dies:

RESTORE-Anweisung

Wörtlich übersetzt bedeutet es: wiederherstellen, restaurieren. Fügen Sie den Befehl vor Zeile 70 ein:

```
69 restore
```

Wenn Sie unser Programmbeispiel jetzt mit RUN starten, werden Sie mit dem Ergebnis zufrieden sein. <RUN/STOP> bricht die Endlosschleife jederzeit ab. Das Listing finden Sie als Programmbeispiel **readdata** auf unserer Diskette.

Noch ein lustiges Experiment zeigt, wie man hier die RND-Funktion einsetzen kann:

```
30 for i=1 to int(rnd(1)*5
)+1
40 read a$
50 next
60 print a$
100 data sabine,monika,ka-
rin,
brigitte,marion
```

Können Sie sich nicht entscheiden, welche der fünf Damen Sie heute abend anrufen sollen? Dann lassen Sie sich mit diesem Programm vom Computer beraten...

Da der PRINT-Befehl erst nach der Schleife eingesetzt wird, bringt er nur den zuletzt gelesenen Namen auf den Bildschirm. Der Computer trifft eine zufällige Auswahl (hoffentlich war's die richtige!). Auf der Diskette heißt das Programm **rndread**.

Mit dem bisher erworbenen Wissen nehmen wir unser erstes größeres Programmprojekt in Angriff. Es stammt aus dem Buch »Computerspiele und Knobeleien« von Rüdiger Baumann. Folgende Aufgaben muß das Programm erfüllen:

- Der Computer hat 14 Sportarten gespeichert.
- Ein Begriff wird jeweils durch den Zufallsgenerator ausgewählt. Der Spieler kennt lediglich die Wortlänge (angezeigt durch Punkte).
- Der Spieler rät einen Buchstaben (erinnert an das Glücksrad in SAT 1).
- Das Programm vergleicht diesen mit allen anderen. Übereinstimmende Zeichen werden an der richtigen Stelle eingesetzt.
- Das wiederholt sich so lange, bis das Wort vollständig ist.
- Zum Abschluß gibt der Spieler das komplette Wort ein.

Das fertige Programm befindet sich unter dem Namen **begriffe raten** auf der Diskette zu diesem Heft. Beachten Sie die Programmerklärung im Infokasten.

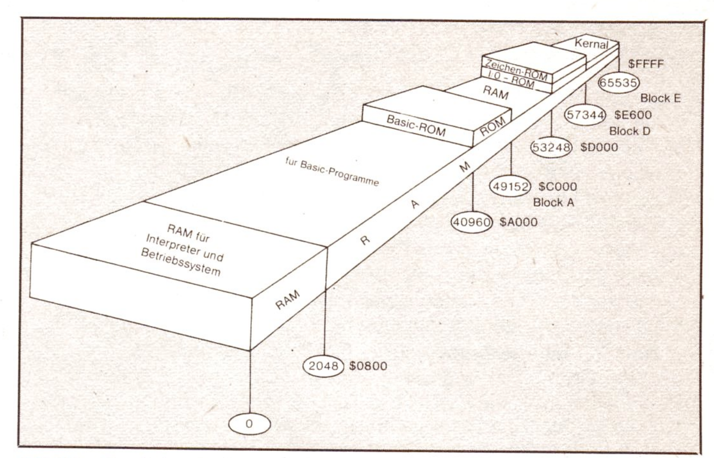
Maschinensprache – in Basic?

Der gesamte Speicher des C64 ist aus einzelnen Zellen aufgebaut, die jeweils Werte von 0 bis 255 enthalten können. Der C64 besitzt insgesamt 65536 solcher Speicherstellen, die von 0 bis 65535 durchnumeriert sind. Sie werden häufiger als Adressen gezeichnet. Um Werte in Speicherstellen einzutragen oder sie auszulesen, ist es notwendig, sich halbwegs in diesem Adreßbuch (Memory Map) des C 64 auszukennen (**Abb.8**):

- Der Speicher beginnt bei Adresse 0.
- Den Bereich der ersten 256 Byte nennt man Zeropage (Nullseite). Diese Adressen braucht der Computer bis auf ganz wenige Ausnahmen selbst.
- Von Adresse 256 bis 511 liegt der Stack (Stapelspeicher), den der C64 ebenfalls alleine verwalten will.
- Der Basic-Eingabepuffer beginnt bei Speicherzelle 512 und endet bei Adresse 600.
- Ab Speicherstelle 601 bis 1023 liegen wichtige Adressen (z.B die Betriebssystemvektoren ab 768), die dem C64 während seiner Arbeit wichtige Werte übermitteln oder interne Berechnungsergebnisse zwischenspeichern. Ausnahmen sind die Bereiche 704 bis 767 (zur freien Verfügung, z.B. Spriteblock 11) und 828 bis 1019 (Kassettenpuffer).
- Der Bildschirmspeicher umfaßt 1000 Adressen und liegt im Bereich von 1024 bis 2023.
- Die Speicherzellen 2040 bis 2047 sind für die Sprite-Zeiger reserviert.
- Endlich: Adresse 2048. Hier beginnt der RAM-Speicher (Random Access Memory: In diese Adressen kann man beliebige Werte zwischen 0 und 255 eintragen) für Basic-Programme. Auch Variablen, Arrays und Zeichenketten sind dort untergebracht.
- Das Basic-RAM endet mit Adresse 40959. Dem Programmierer stehen 38911 Basic-Byte zur freien Verfügung.
- Der Bereich von 40960 bis 49151 ist ROM (Read Only Memory: Speicher, der sich nur lesen, aber nicht beschreiben läßt) gehört dem Basic-Interpreter.
- Die Adressen 49152 bis 53247 bilden ebenfalls einen freien RAM-Bereich, der sich aber von Basic nicht nutzen läßt: Man kann keine Programmzeilen hineinschreiben. Hier werden mit Vorliebe Maschinenspracheprogramme abgelegt.
- Ab 53248 gehört im ROM alles dem Betriebssystem: Manche Bereiche sind sogar mehrfach belegt: VIC-Chip, Ein- und Ausgaberoutinen, Zeichensatz und Floppyoperationen. Auf welche Konfiguration sie zugreifen, regelt der Inhalt in Adresse 1 (Zeropage).

Ein tieferer Einstieg in die Speicherbelegung des C64 gehört aber nicht in einen Basic-, sondern in einen Assembler-Kurs. Uns interessiert nur, wie wir Daten in gewünschte Speicherzellen schreiben, deren Inhalt daraus lesen und bereits vorhandene Systemroutinen oder Maschinenspracheprogramme aktivieren können.

Basic besitzt zur Manipulation des Speichers zwei Befehle: POKE (Schreiben) und PEEK (Lesen). Während der Lesebefehl völlig ungefährlich ist – er kopiert quasi den Inhalt der Speicherzelle und läßt das Original unverändert – kann der Schreibbefehl POKE gefährlich sein: Er verändert



[8] Die Speicherbelegung des C64 auf einen Blick

Kurzinfo: begriffe raten

Der Titelbildschirm gibt die Kurzbeschreibung des Programms aus und stellt sofort die Frage nach dem ersten Begriff. Wie lang er ist, erkennen Sie an der Punkteanzahl. Hinter der INPUT-Abfrage »Was raten Sie?« müssen Sie einen Buchstaben eingeben. Falls er im Wort vorkommt, erscheint er in der Punktezeile an richtiger Stelle.

Zeilen 10 bis 180: Bildschirm löschen und Spielanleitung angeben.

Zeilen 240 bis 260: Per RND-Funktion sucht der Computer aus den Data-Zeilen die Sportart (U$) aus.

Zeile 290: mißt die Länge von U$.

Zeile 300: generiert die Punktekette H$.

Zeile 370: gibt den Punkte-String auf dem Bildschirm aus. Jetzt kennen Sie zumindest die Länge des Suchbegriffs.

Zeile 390: Die INPUT-Routine fordert Sie auf, das Zeichen L$ einzugeben.

Zeile 400: Haben Sie auf Anhieb das richtige Gesamtwort erraten, springt das Programm zu Zeile 600.

Zeile 410: Wenn Sie mehr als **einen** Buchstaben eintippen, wird die Frage wiederholt.

Zeilen 480 bis 530: bilden eine Zählschleife in der Länge des vorgegebenen Begriffs U$. In **Zeile 490** holt das Programm die einzelnen Buchstaben per MID$-Befehl aus U$ heraus, **Zeile 500** macht das gleiche mit H$. Das Programm vergleicht den geratenen Buchstaben L$ mit jedem einzelnen (B$) des Worts U$. Ist er identisch, wird ein neues Hilfswort aus dem richtig geratenen Buchstaben L$ gebildet. War der Ratebuchstabe falsch, übernimmt das Programm aus dem Hilfswort H$ einen Punkt in N$. Das geschieht für alle entsprechenden Stellen der gleich langen Wörter U$ und H$.

Zeile 540: ordnet das neue Hilfswort N$ der Zeichenkette H$ zu, die dem Spieler beim nächsten Ratedurchgang gezeigt wird.

Zeilen 600 bis 670: Dieser Programmteil fragt, ob Sie das Spiel wiederholen oder beenden möchten.

Zeilen 730 bis 760: speichern die Begriffe der Sportarten als String-Datas.

den Inhalt einer Speicherzelle. Man sollte sich vorher über die Folgen klar sein. POKEn Sie eine Zahl in Bereichsadressen, mit denen der C64 seine eigenen Abläufe steuert (sehr kritisch: die Zeropage!), kann's zum Absturz kommen: Es läuft nichts mehr, unter Umständen versagt sogar die Tastenkombination <RUN/STOP RESTORE>. Kaputt machen Sie allerdings nichts damit – das Schlimmste, was passieren kann, ist, daß Sie ein- und ausschalten müssen. Das Programm im Speicher ist dann allerdings futsch. Wir empfehlen: Bevor Sie einen kritischen POKE-Befehl eingeben oder ausprobieren möchten, sichern Sie vorher Ihr Programm!

POKE-Befehl

Damit können Sie beliebige Werte zwischen 0 und 255 in einer Speicherstelle unterbringen. Jede höhere Zahl bringt die Fehlermeldung: Illegal Quantity. Nach dem Befehlswort muß die Adresse und – durch ein Komma getrennt – der Wert stehen. Beispiel: Die Speicherzelle 646 informiert den Computer ständig über die aktuelle Farbe der Schriftzeichen:

```
POKE 646,1
```

färbt die folgenden Zeichen weiß (Farbcode »1«). Der POKE-Wert läßt sich auch in einer Variablen ablegen:

```
F=1: POKE 646,F
```

Unter der Rubrik »Tips & Tricks« in unseren Sonderheften haben Sie sicher schon viele POKE-Adressen gefunden, mit denen man verblüffende Effekte erzielen kann. POKEn wir ruhig ein bißchen weiter:

```
POKE 199,1: PRINT"TEST"
```

bringt das Wort »Test« in reversen Buchstaben,

```
POKE 650,128
```

erlaubt eine Wiederholfunktion aller Tasten,

```
POKE 649,0
```

Nach dieser Eingabe nimmt der C64 keine Eingabe mehr an: Nur noch <RUN/STOP RESTORE> hilft!

```
POKE 53281,0: POKE 53280,7
```

gestaltet den Bildschirmrahmen gelb und den Hintergrund schwarz.

Über diese beiden Adressen läßt sich z.B. überprüfen, ob tatsächlich alle 256 möglichen Werte (von 0 bis 255) darin stehen dürfen:

```
10 for i=0 to 255
20 poke 53281,i
30 print i
40 get a$: if a$="" then 40
50 next i
```

Mit diesem Programm **farbwechsel** (auf Diskette) leiern Sie per Tastendruck (am besten <SPACE>) alle Farben durch. Auf dem Bildschirm erscheint die aktuelle Codenummer. Dabei fällt auf, daß die Farben nach jeweils 16 Durchgängen wiederholt werden: Farbcode 0 (schwarz) ist also identisch mit 16, 32, 48 usw. Wenn Sie den Finger auf der Leertaste lassen, geht's mit dem Programmdurchlauf schneller. Wie man die RND-Funktion einsetzt, zeigt das Programm **rnd-farben**: In rascher Folge nehmen Hintergrund und Rahmen gleichzeitig eine der 16 möglichen Farben an. Das Programm ist kinderleicht zu verstehen.

PEEK-Anweisung

Verwenden Sie diesen Befehl, wenn Sie in eine Speicherstelle hineinschauen möchten:

```
PRINT PEEK(646)
```

gibt – allerdings nur mit Hilfe der PRINT-Anweisung – den aktuellen Inhalt von Adresse 646 auf dem Bildschirm aus. Der PEEK-Wert kann ebenfalls als numerische Variable definiert sein:

```
IN=PEEK(40960):PRINT IN
```

In Verbindung mit einer Schleife lassen sich damit auch ganze Speicherbereiche auslesen und deren Inhalte auf den Bildschirm bringen. Laden Sie z.B. unser kleines Programm **rnd-farben**. Beginn und Ende des Basic-Programms sind als Low-/High-Byte in den Adressen 43, 44 (= Anfang, normalerweise immer »2049«) und 45, 46 (= Ende) gespeichert. Erledigen Sie folgende Eingaben im Direktmodus (die FOR-NEXT-Schleife muß in einer Zeile stehen):

```
BE=2049
EN=PEEK(45)+256*PEEK(46)-1
FOR I=BE TO EN:
PRINTPEEK(I);: NEXT
```

Der Bildschirm füllt sich mit einer Menge Zahlen: Die Speicherstellen des Basic-Programms haben ihren Inhalt offenbart. So liest sie auch der Computer. Die drei Nullen am Schluß weisen daraufhin, daß das Basic-Programm durchlaufen ist. Dieses Prinzip verfolgen die meisten »Data-Maker«-Programme, die aus bestimmten Speicherbereichen per PEEK Inhalte auslesen (überwiegend aus Maschinenspracheprogammen) und sie durch geschickte Programmierung in – für Basic lesbare – DATA-Zeilen umwandeln. Das Basic-Programm erhält als Kopfzeile eine FOR-NEXT-Schleife, die nach dem Laden die Datas liest (per READ) und mit POKE in die gewünschten Speicherbereiche bringt. Diese Art, Maschinenprogramme an ihren angestammten Platz zu schieben, nennt man Basic- oder DATA-Lader. Beispiel: das Utilty **goto x** auf der Diskette zum Sonderheft.

Bildschirm- und Farb-RAM

Die Wirkung eines POKEs läßt sich nirgends so eindrucksvoll demonstrieren wie im Bildschirmspeicher (1024 bis 2023). Kleines Experiment gefällig?
- Löschen Sie mit <SHIFT CLR/HOME> den Bildschirm,
- Bringen Sie den Cursor an den unteren Bildschirmrand,
- Geben Sie ein:

```
POKE 1024,42
```

In der linken oberen Ecke ist plötzlich ein Sternchen aufgetaucht. Machen Sie aus »42« die Zahl »43«, erscheint das Plus-Zeichen. Ändern Sie die Adreßangabe 1024 in einen beliebigen Wert bis maximal 2023: An entsprechender Stelle auf dem Bildschirm erscheint das Zeichen, das mit der Codezahl hinter dem Komma eingestellt wurde. Eine Übersicht der Bildschirmcodes finden Sie im Handbuch zum C64. Die Bildschirm-POKEs sind eine Alternative zum PRINT-Befehl:

```
10 for i=0 to 4
20 read a
30 poke1024 + i,a
40 next
100 data 8,1,12,12,15
```

macht das gleiche wie:

```
10 printchr$(147)"hallo"
```

Beim direkten POKEn in den Bildschirm entfallen Cursor-Steuerzeichen: Das Zeichen wird immer in die durch die POKE-Adresse bestimmte Position gebracht. Man kann damit Effekte erzielen, die durch den PRINT-Befehl nur sehr umständlich zu realisieren sind. Unser Beispiel soll eine bewegte, farbige Bildschirmumrandung erzeugen. Dazu gehen wir Schritt für Schritt vor:

```
10 for i=0 to 999
20 poke 1024+i,42
30 next
```

Alle 1000 Speicherstellen des Bildschirm-RAM werden per Schleife mit einem Stern besetzt. Es fällt auf, daß die READY-Meldung und der Cursor nicht hinter dem letzten Stern (Adresse 2023) erscheinen, sondern weiter oben: dort, wo der letzte Basic-Befehl (RUN) auf den Bildschirm geschrieben wurde. Im nächsten Schritt sollen die Sterne nur auf der obersten und untersten Zeile erscheinen. Unsere Schleife zählt jetzt lediglich bis 39:

```
10 for i=0 to 39
20 poke 1024+i,42
30 poke 1024+960+i,42
40 next i
```

Neu ist Zeile 30: Sie erhöht die POKE-Adresse um 24 x 40 = 960 Plätze. Das entspricht der untersten Bildschirmzeile.

Nun fehlen uns noch der linke und rechte Rand:

```
50 for k=0 to 960 step 40
60 poke 1024+k,42
70 poke 1024+39+k,42
80 next k
90 goto 10
```

Um von oben nach unten zu zählen, beginnen wir mit »0«, gehen aber in 40er-Schritten immer an den Anfang der nächsten Bildschirmzeile: Damit wird der linke Rand erzeugt. Zeile 70 benutzt für den rechten Rand die gleiche Art des Zählens, POKEt aber den Stern um 39 Plätze verschoben: die äußerste, rechte Spalte. Zeile 90 erzeugt eine Endlosschleife, um das lästige READY zu verhindern.

Nun schütten wir Farbe drüber: Nach jedem Umlauf soll die Umrandung in einer anderen Farbe erscheinen. Dazu bemühen wir das für den Bildschirmspeicher im Textmodus zuständige Farb-RAM (im Bereich von 55296 bis 56295). Es sind ebenfalls 1000 Speicherstellen: Jede Adresse des Farb-RAM korrespondiert mit der entsprechenden des Bildschirms:

Adresse Bildschirm-RAM = Adresse Farb-RAM - 54272

Das Ergebnis unserer gemeinsamen Anstrengungen sehen Sie in der Datei **farbrand** auf der Diskette zum Sonderheft. Die entsprechende Beschreibung steht im Infokasten.

Kurzinfo: farbrand

Nach dem Start mit RUN versieht der Computer den Bildschirmrand mit Sternen, die bei jedem Umlauf die Farbe wechseln.

Zeilen 5 und 90: bilden die übergeordnete Schleife F. Dadurch kommen die Farbwerte ins Farb-RAM.

Zeilen 10 und 40: öffnen und regulieren die innere Schleife I.

Zeile 20: schreibt das Sternchen an die durch den Zähler I definierte Bildschirmposition (erste Zeile). Die dazugehörige Farbspeicherstelle wird mit dem aktuellen Wert F belegt.

Zeile 30: wie Zeile 20, nur gilt das jetzt für die unterste Bildschirmzeile.

Zeilen 50 bis 80: kümmern sich in einer weiteren Schleife K um den linken und rechten Bildschirmrand.

Das Programm **farbumlauf** ist eine Variante von **farbrand:** Die Umrandung soll nicht oben und unten bzw. links und rechts gleichzeitig verlaufen, sondern im Kreis! Das erreichen Sie, indem Sie für jede POKE-Zeile eine eigene Schleife aufmachen: in den Zeilen 30 und 60 wird mit negativem STEP sogar rückwärts gezählt.

Mit dem POKE-Befehl und einer Schleife lassen sich bei Basic-Ladern beliebige Speicherbereiche mit Werten ausstatten, die in DATA-Zeilen abgelegt sind. Ausnahme: Der Bereich im Basic-RAM, den der DATA-Lader selbst beansprucht. Am bekanntesten sind die POKEs in Speicherbereiche, in denen sich mit Vorliebe Maschinenspracheprogramme tummeln: die Adressen 49152 bis 53247 und im Kassettenpuffer (828 bis 1019). Assembler-Dateien lassen sich allerdings nicht mit RUN starten. Dazu benötigt man einen speziellen Basic-Befehl:

SYS-Anweisung

Dahinter muß die Einsprungadresse stehen (das erste Byte, das die Maschinenroutine aufruft). Wichtig ist, daß am Ende der mit SYS gestarteten Routine ein RTS-Befehl steht (Code $60 hex., 96 dez.). Dadurch kehrt man aus der Maschinenroutine wieder in den übergeordneten Basic-Interpreter zurück (der ja auch nichts anderes ist als ein großes Maschinenprogramm!). Selbstverständlich kann man Teile daraus (Unterprogramme) ebenfalls mit einer SYS-Anweisung aufrufen und im eigenen Basic-Programm verwenden. Es ist erstaunlich, wie viele interessante Routinen bereits im Betriebssystem enthalten sind, die sich per SYS initialisieren lassen: Oft machen sich Basic-Programmierer unnötig die Mühe, denselben Effekt durch umständliche Basic-Programmierung zu erzeugen. Beispiele gibt's genug:

1. In der obersten und untersten Bildschirmzeile soll Text ausgegeben werden. Die Mehrzahl der Basic-Programmierer arbeitet hier mit dem inversen <s> = Taste <HOME> und 24 reversen Steuerzeichen <q> = Taste <CRSR abwärts>. Ganz zu schweigen davon, daß unter zu häufiger Verwendung inverser Steuerzeichen die Übersichtlichkeit eines Programmlistings leidet. Einfacher geht's mit zwei POKEs und einem SYS-Befehl:

```
POKE 214, Zeile
POKE 211, Spalte
SYS 58640
```

Einen Kaltstart des Computers (Ausgangslage nach dem Einschalten) erzielen Sie mit diesen Anweisungen:

```
SYS 58260
```

oder

```
SYS 64738
```

War ein Basic-Programm im Speicher, ist es jetzt natürlich gelöscht!

Die LOAD-Routine z.B. läßt sich mit

```
SYS 62622
```

allerdings nicht starten. Es fehlen die nötigen Parameter: logische Filenummer, Geräte- und Sekundäradresse, Filename, Anfangs- und Endadresse usw. Die Angaben sucht sich der Basic-Interpreter nach der Eingabe von LOAD selbst in den entsprechenden Speicherstellen und den von Ihnen übergebenen Parametern zusammen. Beispiel:

```
LOAD "TEST",8
```

teilt dem Computer mit, daß er die Datei mit dem Namen »Test« vom Gerät mit der Nr. 8 (Floppy) laden soll.

Tippen Sie nämlich nur »LOAD«, erscheint die Meldung:

```
PRESS PLAY ON TAPE
```

Da der C64 keinen Dateinamen und keine Geräteadresse findet, nimmt er an, Sie möchten von Datasette laden. Das geschieht auch, wenn Sie »SYS 57704« eingeben: Dort beginnt die Betriebssystemroutine LOAD im Basic-Interpreter. Alle Parameter, die nicht die Datasette als Ladegerät kennzeichnen, müssen Sie von Hand eingeben. Dazu besitzt der Computer vier wichtige Speicherstellen, die jedem Assembler-Programmierer geläufig sind (ohne diese Register wäre er aufgeschmissen!):

Akkumulator (A): 780,
X-Register (X): 781,
Y-Register (Y): 782 und
P-Register (P): 783.

Das P-Register, häufiger als Statusregister des Prozessors bezeichnet, ist nicht so einfach zu handhaben wie die anderen. Es besitzt bei allen 8 Bit lediglich eine Schalterfunktion: Bit an = 1, Bit aus = 0. Jedes Bit des Status-Bytes kümmert sich um einen bestimmten Computerzustand (**Tabelle 3**). Ist eines der Bits eingeschaltet (= 1), heißt es für den Computer – bildlich gesehen – eine Flagge (Flag). Assembler-Programmierer... müssen die Flaggen stets im Auge behalten, für Basic-Programmierer hat das Byte 783 so gut wie keine Bedeutung.

Tabelle 3. Prozessorstatusregister 783 (Funktionen)

Bit	Wert	wenn Bit = 1	Abkürzung
0	1	Übertrag	C(arry)
1	2	Null	Z(ero)
2	4	Unterbrechung	(I)nterrupt
3	8	Dezimal	D
4	16	Abbruch	B(reak)
5	32	unbenutzt	
6	64	Überlauf	O(V)erflow
7	128	Vorzeichen	N(egativ)

Die Routine PLOT ($FFF0, 65520) im Betriebssystem beispielsweise benützt zwei der genannten Register, um den Cursor beliebig auf dem Bildschirm zu positionieren: das X- und Y-Register. Die bereits erwähnte Anweisung zur Cursorpositionierung könnte auch so lauten:

```
POKE 781,Zeile
POKE 782,Spalte
SYS 65520
```

Systemroutinen nutzen

Weitere SYS-Anweisungen, die Betriebssystemroutinen aktivieren:
SYS 58692: löscht den Bildschirm,
SYS 58726: bringt den Cursor in die <HOME>-Position (linke obere Bildschirmecke),
SYS 59903: löscht eine Zeile auf dem Bildschirm. Im X-Register (781) muß allerdings die Zeilennummer stehen. Dazu ein Programmbeispiel:

```
10 sys 58692
20 for i=1 to 22
30 print"das ist ein text."
40 next
50 for zl=0 to 22step2
60 poke 783,peek(783)and254
70 poke 781,zl: sys 59903
80 next
```

Nach dem Start mit RUN gibt der Computer 22 Bildschirmzeilen mit Text aus, anschließend wird durch die Zeilen 50 bis 70 jede zweite Textzeile gelöscht. Erwähnenswert: Vor jedem Aufruf der PLOT-Routine 65520 muß das Carry-Flag gelöscht sein (Bit Nr. 0 in Adresse 783). Dies erledigt die logische UND-Verknüpfung in Zeile 60. Ein weiteres Programmbeispiel, das fast alle bisher erwähnten SYS-Befehle vereint, finden Sie auf der Diskette unter **zeilenkiller**.

Auch die interne Zeit des C64 läßt sich mit einer SYS-Anweisung auslesen: 65502 – zumindest in 1/60tel-Sekunden (wie die Systemvariable TI). Nachdem die Betriebssystemroutine ihre Arbeit getan hat, haben sich drei Speicherstellen verändert: 780, 781 und 782. Sie müssen lediglich passend zusammengefügt werden, um die richtige Zeit zu erhalten:

```
PRINT PEEK(780) +
PEEK(781) * 256 +
PEEK(782)
```

Das Ergebnis ist bis auf eine kleine Abweichung identisch mit dem aktuellen Wert von TI. Den Zeitunterschied verbraucht der Basic-Interpreter, um Ihre Eingaben zu übersetzen, auszuführen und auf den Bildschirm zu bringen: deutlich zu sehen in unserem Beispielprogramm **sys-zeit** auf der Diskette zum Sonderheft. Achtung: Übersteigt der Inhalt von 781 den Wert »255«, wird er automatisch wieder auf »0« gesetzt.

Der nächste Befehl ist das Mauerblümchen unter den Befehlen des Basic 2.0, obwohl er mehr kann als SYS:

USR-Befehl

In der Funktionsweise ist diese Anweisung mit SYS identisch: Sie muß eine Routine in Maschinensprache aus einem Basic-Programm aufrufen. Aber die Befehlssyntax unterscheidet sich erheblich: Steht bei SYS die Einsprungadresse unmittelbar hinter dem Befehl, muß man diese bei USR zunächst in Low- und High-Byte splitten und in die Adressen 785/786 POKEn. Ein Beispiel:

Sprung zu Adresse 64738 (Kaltstart/Reset)

```
10 POKE 785,64738 - 256 *
   INT(64738/256)
20 POKE 786,INT(64738/256)
30 PRINT USR(0)
```

Das kurze Beispiel ist ebenfalls auf der Diskette zum Sonderheft: **usr(0)**.

Der Vorteil von USR: Der Befehl selbst ist eine numerische Variable und läßt sich in jede andere umbenennen (z.B. X, A, B usw). Statt »0« kann ebenfalls eine numerische Variable (z.B Y) in Klammern stehen, die man im Programmverlauf auch berechnen lassen darf. Das Argument Y kommt in die Speicherzellen 97 bis 102 im Gleitpunktakkumulator FAC 1. Als Fließkommazahl wird es nun vom aktivierten Maschinenprogramm weiterverarbeitet. Das Resultat kehrt zurück in FAC 1 und steht dann als Wert X zur Verfügung. Y kann auch ein komplexer Ausdruck sein, z.B.:

```
X = USR(PEEK(A)+256*PEEK(B-
))
```

Experimentieren wir mit diesem Befehl: Wir möchten eine Rechenroutine im Betriebssystem des C64 aufrufen, die für die INT-Funktion zuständig ist: 48332. Zuerst weisen wir der Variablen Y eine Realzahl zu:

```
10 y = 365.57
```

Jetzt POKEn wir Low- und High-Byte von 48332 in die Adressen 785 und 786:

Low-Byte von 48332 = 48332 - 256 * INT (48332/256) = 204
High-Byte von 48332 = INT(48332/256) = 188.

```
20 poke 785,204
30 poke 786,188
```

Fehlt nur noch der USR-Befehl und die Ausgabe des Resultats:

```
40 x = usr(y)
50 print x
```

Tippen Sie das kurze Programm ab und starten Sie es mit RUN. Als Ergebnis erhalten Sie die Integerzahl »365«.

Ist Ihnen aufgefallen, daß wir in unserer Serie der 780er Speicherstellen (Akkumulator, x-, y- und P-Register, USR-Adressen) die Adresse 784 überhaupt noch nicht erwähnt haben? Sie besitzt den Wert »76« ($4C hex.), das ist der Code für den Assembler-Befehl JMP. Wird die USR-Anweisung ausgeführt, springt der Computer automatisch zu dieser Speicherstelle und holt sich aus den folgenden beiden Bytes 785/786 die Adresse, wohin er springen soll: Die Ähnlichkeit mit der SYS-Anweisung ist nicht zu übersehen. Besonders, wenn in Ihren Basic-Programmen extrem schnelle Unterroutinen in Maschinensprache eingebaut sind (z.B. Joystick-Abfragen), läßt sich USR elegant anwenden. Denken Sie daran, daß man damit Variablenwerte aus Maschinenspracheprogrammen an Basic weitergeben kann!

Kurzinfo: zeilenkiller

Mit der Eingabe von RUN bringt der Bildschirm 22 Textzeilen. Nach 1 Sekunde werden die Zeilen 7 bis 15 gelöscht, nach einer weiteren Sekunde dort wieder eingefügt. Das Programm befindet sich in einer Endlosschleife und läßt sich mit <RUN/STOP> abbrechen.

Zeile 10: löscht den Bildschirm (ersetzt das inverse Zeichen von <SHIFT CLR/HOME> bzw. CHR$(147).

Zeilen 20 bis 40: In der Schleife I wird 22mal das Unterprogramm in Zeile 1000 angesprungen (Ausgabe der Textzeile).

Zeile 45: Aufruf des Unterprogramm-Einzeilers »Warte ca. 1 s« in Zeile 1100!

Zeilen 50 bis 70: Der jeweilige Variablenwert ZL wird in Speicherstelle 781 (x-Reg.) gePOKEt und die Löschroutine aktiviert.

Zeilen 80 bis 110: Die leeren Bildschirmzeilen 7 bis 15 sollen wieder gefüllt werden: Zuerst muß man sicherheitshalber in Adresse 783 (Statusregister) das Carry-Bit löschen. In Speicherstelle 781 kommt die aktuelle Zeile, in 782 (y-Reg.) die aktuelle Spalte (ist immer »0« = Zeilenanfang). Mit dem anschließenden SYS-Befehl aktiviert man die PLOT-Routine und schreibt den Text wieder in die Lücke auf dem Bildschirm.

Zeilen 120 bis 130: bringt den Cursor in HOME-Position, der nächste Durchgang beginnt...

Der goldene Mittelweg

Basic und Assembler sind zwei Paar Stiefel: trotzdem können sich beide Sprachen ideal ergänzen, wenn Sie nicht nur selbstentwickelte Maschinenroutinen mit Basic-Programmen verwenden, sondern die bereits vorhandenen Assembler-Routinen im Betriebssystem schamlos ausnutzen. Einige dieser Unterprogramme, die per SYS oder USR(X) initialisiert werden, haben wir bereits kennengelernt.

Eine Aufgabe, die der Computer häufig zu erfüllen hat, ist das Verschieben von Speicherbereichen. Möchten Sie z.B. den Originalzeichensatz des C64 ändern, müssen Sie ihn vom ROM ins RAM kopieren. Dort können Sie aus den Zeichen z.B. Pflastersteine machen oder deutsche Umlaute und Sonderzeichen fabrizieren. Ein weiteres Beispiel ist das Kopieren des Basic- oder Betriebssystem-ROM ins darunterliegende RAM, um dort irgendwelche Maschinenspracheroutinen oder

Texttabellen zu verändern (z.B. Fehlermeldungen eindeutschen). Dies geht in Basic problemlos, z.B. kopieren Sie so das Basic-ROM ins RAM:

```
10 for i=40960 to 49151:
    pokei,peek(i): next
```

Da die Ausführung dieser Basic-Zeile bei 8192 Byte unerträglich lange dauert (35 s), greifen wir – ebenfalls von Basic aus – auf eine Routine zurück, die uns der Interpreter zur Verfügung stellt: Die erledigt das in Sekundenschnelle! Sie liegt bei $A3BF (41919) und verlangt als Low- und Highbyte in den Speicherstellen 95/96 die Startadresse des Bereichs, der übertragen werden soll. In den Speicherzellen 90/91 muß dagegen die Endadresse + 1 stehen und in den Zellen 88/89 befindet sich das Ende des neuen Bereichs, ebenfalls um »1« erhöht. Wem's zu kompliziert wird, der sollte das Beispielprogramm **blockmove** von unserer Diskette laden und die Zahlen eingeben, die im Basic-Einzeiler stehen:

- Alte Startadresse: 40960
- Alte Endadresse + 1: 49152
- Neue Endadresse + 1: 49152

Gerade Maschinensprachеprogramme werden oft mit der Endung »,8,1« geladen. Das bedeutet, daß der Computer die vor der eigentlichen Programmdatei auf Diskette gespeicherte Ladeadresse verwendet. Diese Startadresse wird beim SAVEn bereits auf Diskette abgelegt, bei Basic-Programmen lautet sie immer: 2049. Solche »absoluten« Dateien müssen nicht immer Programme sein: Jeder beliebige Speicherbereich des C64 läßt sich auf Diskette auslagern – man muß nur wissen, wie. Mit SAVE? Dieser Befehl funktioniert lediglich bei Basic-Programmen im Basic-RAM. Sie können damit z.B. kein Maschinensprache-Utility speichern, das sich ab 49152 ($C000) im Computer befindet. Dafür ist wieder eine Betriebssystemroutine mit den entsprechenden Parametern zuständig: Sie liegt im ROM bei Adresse $F5ED (62957). Allerdings bleiben uns als Basic-Programmierer auch hier gewisse Vorbereitungen nicht erspart, die auch ein Assembler-Freak treffen muß: Mit SYS (57812) ruft man eine Routine auf, die bestimmte Parameter für LOAD oder SAVE aus dem Basic-Text liest. Es darf kein Komma folgen! Unmittelbar dahinter schließt sich der Filename an (im Klartext mit Anführungszeichen oder als Stringvariable). Jetzt muß allerdings ein Komma gesetzt werden, denn nun kommt die Geräteadresse (8). Die Start- und Endadresse wird per INPUT-Abfrage in den numerischen Variablen S und E gespeichert. Nach Zerlegung in Low- und High-Byte kommt die Startzahl in die Adressen 193/194, das Ende in die Speicherstellen 174/175. Der abschließende SYS-Befehl 62957 aktiviert die Speicherroutine und legt den angegebenen Bereich – Bit für Bit – auf einer Diskette ab. Das kann z.B. ein Textbildschirm sein:

- Anfangsadresse: 1024
- Endadresse + 1: 2024

oder eine Hires-Grafik:

- Anfangsadresse: 8192
- Endadresse + 1: 16384

oder Sprite-Daten (z.B. in Block 11):

- Anfangsadresse: 704
- Endadresse + 1: 768

Diese Liste könnte man beliebig fortsetzen: Das kurze Utility **sys-save** speichert wirklich jeden Bereich Ihres Computers!

Operation Daten-File

Basic-Programme bestehen aus einer Sammlung numerischer Zeilen, die Anweisungen an den Computer enthalten. Daten, die dem Programm sofort zur Verfügung stehen sollen, müssen als String- oder Zahlenvarialben fest enthalten sein – ebenfalls in Programmzeilen abgelegt.

Eine Datei (File) dagegen besteht nur aus reinen Einträgen (wie z.B. ein Telefonbuch), ohne Hinweis, was damit gemacht werden soll. Richard Mansfield, ein bekannter Computerjournalist, hat den Unterschied zwischen Programm und Datei anhand eines einleuchtenden Beispiels erklärt:

Stellen Sie sich vor, Sie haben im Supermarkt ein Dose Tomatensuppe gekauft. Der Dosenaufkleber enthält beides: Programm und File.

Das ist die Datei: Wasser, Tomaten, Salz, Monosodiumglutamat, Farbstoff, Oleoresin.

Das Programm dazu heißt:

1. Vorsichtig öffnen
2. Inhalt in einen Topf gießen
3. eine Tasse Wasser hinzufügen
4. auf kleiner Flamme erhitzen
5. ungewürzt servieren

Programme speichert man mit SAVE. Geladen werden Sie mit LOAD.

Dateien benötigen eine eigene Befehlskombination mehrerer Basic-Wörter.

OPEN-Anweisung

Will man eine Datei laden oder speichern, muß zunächst eine Verbindung vom Computer zum externen Speichergerät (Floppy) hergestellt werden: eine Anweisung an den C64, sich für den Datentransfer bereitzuhalten. Man nennt dies »eine Datei öffnen«, vergleichbar mit dem Öffnen eines Karteikastens.

Die zusätzlichen Angaben (Parameter) hinter OPEN erfüllen wichtige Aufgaben:

Dateinummer

Damit wird die geöffnete Datei gekennzeichnet. Erlaubt sind Zahlen zwischen 1 und 255 (in unserem Beispiel »2«). Die Dateinummer dient als Referenz bei jedem Zugriff, um den Computer nicht zu verwirren – es können immerhin bis zu zehn verschiedene Dateien geöffnet sein.

Gerätenummer

...ist die Kennzahl dafür, mit welchem externen Gerät der Computer eine Verbindung herstellen soll:

0: Tastatur
1: Datasette
3: Bildschirm
4: oder 5: Drucker
6: Plotter
8 bis 11: Diskettenstationen

Fehlt die Angabe der Gerätenummer, setzt der C64 automatisch »1« ein (Datasette).

Sekundäradresse

...kann ebenfalls eine Zahl zwischen 0 und 255 sein. Bei den verschiedenen Peripheriegeräten hat sie unterschiedliche Bedeutung:

Diskettenlaufwerk:
0 und 1: reserviert vom Betriebssystem
2 bis 14: stellt einen Datenkanal zur Verfügung. Achtung: Es können nur maximal drei Datenkanäle **gleichzeitig** geöffnet sein!
15: reserviert für den Fehler- bzw. Befehlskanal

Dateiname

Diese Zeichenkette wird als Name ins Directory der Floppy übernommen. Sie darf auch eine Stringvariable sein, im Klartext muß sie allerdings in Anführungszeichen stehen.

Dateityp

Man benötigt ihn nur bei Diskettenoperationen. Es genügt, den ersten Buchstaben anzugeben:

SEQ (S): sequentielle Datei
USR (U): User-Datei
PRG (P): Programmdatei
REL (R): Relative Datei

SEQ-Dateien werden am häufigsten verwendet und sind auch am leichtesten zu verstehen. Wir werden uns daher den anderen Dateitypen im Rahmen unseres Kurses nicht widmen.

Modus

...ist ebenfalls nur bei Diskettenoperationen relevant und bestimmt wie der Datenkanal genutzt wird:

W: Datei schreiben (WRITE)
R: Daten lesen (READ)
A: Datei erweitern, neue Daten anfügen (APPEND)

Beginnen wir unsere Dateiarbeit:

```
10 open 2,8,2,"datei,s,w"
```

Es ist eine sequentielle Datei (S), die auf Diskette geschrieben (W) werden soll. Wollen Sie den Dateinamen als Variable definieren (z.B. DT$=»DATEI«), lauten die Befehlszeilen:

```
5 dt$="datei"
10 open 2,8,2,dt$+",s,w"
```

Um jetzt Daten auf die Diskette zu speichern, müssen Sie eine Variation des PRINT-Befehls verwenden:

PRINT#-Anweisung

Dahinter muß unbedingt dieselbe Dateinummer stehen, die Sie bei der OPEN-Anweisung verwendet haben. Dann folgt die Zeichenkette oder Variable, die Sie speichern möchten:

```
20 print#2,"fussball"
30 print#2,"tennis"
40 print#2,"handball"
```

Diese drei Einträge reichen zunächst. Nun soll die Datei wieder geschlossen werden:

CLOSE-Befehl

Hier genügt es, wenn Sie nur die Dateinummer angegeben. Senden Sie sicherheitshalber aber zur exakten Kennzeichnung des Dateiendes zuvor noch einen PRINT#-Befehl mit einem selten benützten Zeichen, z.B. den Stern <*> (warum das so wichtig ist, werden wir noch sehen):

```
50 print#2,"*"
60 close 2
70 end
```

Wenn Sie das Listing (**write.seq** auf der Diskette zum Sonderheft) mit RUN starten, wird die Floppy tätig. Im Disketten-Directory findet man nun eine neue Datei vom Typ SEQ, mit dem Namen »datei«.

Files auf Diskette sind nur dann interessant, wenn man sie auch wieder lesen kann. Das Programm dazu ist in der Struktur identisch mit der Routine fürs Speichern, ein neuer Befehl kommt dazu:

INPUT#-Anweisung

Hiermit holt man Strings und Zeichenketten aus einer Datei wieder in den Computer. Zuvor muß man das File wieder öffnen:

```
10 open2,8,2,"datei,s,r"
```

Die Namen der Sportarten (unsere Daten) wurden vorher im Klartext gespeichert – das geht bei INPUT# nicht! Der Computer kann beim Laden weder wissen, welcher String gemeint noch wie lang er ist (das ist wichtig für die Speicherverwaltung!). Weisen Sie den jeweiligen Zeichenketten, die Sie beim Laden erwarten, beliebige String-Namen zu, z.B. A$, B$ und C$.

```
20 input#2,a$
30 input#2,b$
40 input#2,c$
50 close 2
55 printa$,b$,c$
60 end
```

Falls Sie auch bereits beim Speicher Variablen verwendet haben, ist es unnötig, daß die String-Namen übereinstimmen. Wurde bei PRINT# z.B. X$, Y$ und Z$ benutzt, dürfen die Variablen bei INPUT# getrost E$, F$ und G$ heißen – wichtig ist allein der Variablentyp (numerisch oder Zeichenkette). Sonst gibt's einen »File Data Error«. Das Programm ist ebenfalls auf der Sonderheftdiskette: **read.seq**.

Natürlich lassen sich auch numerische Variablen oder Zahlen speichern. Immer dran denken: Beim Einlesen mit INPUT# dann ebenfalls numerische Variablen verwenden!

Daten kann man noch mit einem anderen Befehl holen:

GET#-Anweisung

Dieser Befehl arbeitet im Prinzip wie INPUT#, liest aber immer nur ein einziges Zeichen! Man verwendet ihn deshalb am besten in einer Schleife:

```
10 open 2,8,2,"datei,s,r"
20 get#2,a$
30 printa$;
40 goto 20
```

Das Programm funktioniert zwar (Sie erhalten die drei Wörter untereinander auf dem Bildschirm), aber die Schleife macht unverdrossen weiter! Jetzt wird klar, warum wir bei der Programmerläuterung zu **write.seq** vorgeschlagen haben, mit dem Stern ein zusätzliches Dateiende-Kennzeichen zu setzen. Dieses Grafikzeichen können wir jetzt nämlich abfragen, wenn Sie folgende Zeilen in unser Beispielprogramm einfügen:

```
21 ifa$=chr$(42) then 50
50 close 2
```

Kurzinfo: literatur-datei

Sie können mit diesem Beispielprogramm maximal 100 Bücher in Datensätzen zu je sechs Datenfeldern erfassen. Der Datenfeldname heißt im Programm immer »Kategorie«. Wir schlagen vor, den Datenfeldern folgende Funktionen zuzuordnen:

1. **Kategorie:** Buchtitel
2. **Kategorie:** Autor
3. **Kategorie:** Verlag
4. **Kategorie:** Sachgebiet
5. **Kategorie:** Preis
6. **Kategorie:** Bestellnr.

Nach diesem Prinzip sind die Datensätze in der sequentiellen Datei »literatur« abgelegt. Selbstverständlich steht dieses universelle Programm auch für andere Dateiarten offen, z.B. Adressen, Videos, Schallplatten usw. Sie können die Kategorienamen Ihren Wünschen entsprechend vergeben.

Zeile 8: Die gesamte Datei wird durch die DIM-Anweisung mit 100 Datensätzen zu je sechs Feldern eingerichtet. Diese Zahlen lassen sich jederzeit verändern und anpassen.

Zeile 9: stellt die Bildschirmfarben ein (Rahmen und Hintergrund schwarz, Schriftfarbe hellgrau).

Zeile 10: löscht den Bildschirm, aktiviert mit CHR$(142) den Großschriftzeichensatz und setzt die Flag-Variable auf »0«.

Zeilen 15 bis 50: Dieser Menütext erscheint auf dem Bildschirm.

Zeilen 55 bis 66: fragen die Tastatur ab, ob eine der Zahlentasten <1> bis <5> gedrückt wurde. Trifft dies zu, springt das Programm zur entsprechenden Unterroutine. Die Rücksprungschleife mußte zweigeteilt werden: Je nach Inhalt des Variablen-Flags FL hüpft das Programm nur zur GET-Abfrage oder zum Menübeginn. In der Beschreibung zu **on.ziffer** haben wir die Gründe für den Einsatz dieses Flags ausführlich erläutert.

Zeilen 100 bis 195: bilden das Unterprogramm »Datei laden«, das man mit der Taste <1> aktiviert. Die Datei »literatur« wird mit der Schleife K und per GOTO-Anweisung in Zeile 170 geladen. Als Datenende-Kennzeichen gilt der Klammeraffe.

Zeilen 200 bis 340: umfassen die Suchroutine, in der nach einem markanten Stichwort gefragt wird. Welche der sechs Kategorien Sie verwenden, ist egal: Nur muß es immer mit dem entsprechenden Datenfeldeintrag exakt übereinstimmen (z.B. 4. Kategorie: Grafik), sonst findet der Computer nichts. Die Suche beginnt mit den beiden Schleifen S und K in den Zeilen 220 und 230. Mit IF-THEN vergleicht der C64 in Zeile 240 das Stichwort (S$) mit den vorliegenden Datenfeldern. Wurde er fündig, bringt er in der Schleife Z alle sechs Datenfelder des dazugehörenden Datensatzes auf den Bildschirm. Die Zeilen 250 und 270 arbeiten die Schleifen K und S ab. Ist der Durchlauf beendet, erscheint die entsprechende Meldung (auch wenn der Computer nichts gefunden hat). Mit <J> können Sie es noch einmal versuchen (zurück zu Zeile 10) oder mit <N> per RETURN-Anweisung ins Menü zurückkehren – nicht, ohne vorher das Flag FL gesetzt zu haben (= 1)!

Zeilen 400 bis 480: kümmern sich um den Menüpunkt 3 (Daten eingeben). Die Zählvariable K erhält den Grundwert 1. Die Zeilen 430 und 440 bilden eine Schleife, die mit GOTO arbeitet. Nacheinander gibt der Computer die sechs Kategoriefragen aus und erwartet Ihre Einträge. Ist die Variable K kleiner als 6, wird sie um »1« erhöht. Das Programm springt zurück zu Zeile 430 und bringt die nächste INPUT-Abfrage. Nach sechs Durchgängen können Sie entscheiden, ob Sie einen weiteren Datensatz (das nächste Buch) erfassen möchten – dann muß die Datensatzmenge D ebenfalls um »1« erhöht werden – oder zum Menü zurückkehren wollen. Vorher ruft der Computer mit GOSUB 501 in Zeile 480 die Speicherroutine auf.

Zeilen 500 bis 580: Es erscheint die Sicherheitsabfrage, ob eine Diskette im Laufwerk liegt. Durch Tipp auf eine beliebige Taste (darauf wartet die Zeile 503) startet das Speichern in Zeile 510 mit dem Öffnen der Datei »literatur« zum Schreiben (S,W). Wer mit dem Klammeraffen vor dem Dateinamen nichts anfangen kann: Damit weisen Sie die Floppy darauf hin, daß eine bereits bestehende Datei auf Diskette überschrieben werden soll (REPLACE-Funktion, bitte im Floppy-Handbuch nachlesen!). Da der Dateiname bei jeder Computersitzung »literatur« lautet, würden Sie ohne den REPLACE-Klammeraffen stets die Fehlermeldung »File exists (Datei existiert bereits auf Diskette)« erhalten. In den Zeilen 520 und 530 beginnt das Speichern mit: D ist die Menge der eingegebenen Datensätze (bei »1« beginnend), K sagt dem Computer, daß zu jedem D immer sechs Datenfelder gehören. Zeile 570 schreibt einen Klammeraffen ans Ende der Gesamtdatei. Geschlossen wird sie in Zeile 570. Anschließend kehrt das Programm wieder ins Hauptmenü zurück (Zeile 580). Achtung: FL muß »1« sein!

Zeilen 600 bis 610: sind fürs Programmende zuständig, wenn man die Taste <5> gedrückt hat: Der Bildschirm wird gelöscht und die READY-Meldung ausgegeben.

GET# liest so lange, bis es den Stern findet und beendet die Leseroutine!

Einen Schönheitsfehler sollten wir noch ausmerzen: Bei **write.seq** stört, daß alle Daten mit einem eigenen PRINT#-Befehl verschickt werden. Bei drei Wörtern geht's noch, aber bei 50 oder 100? Jetzt können Sie unter Beweis stellen, daß Sie unseren Basic-Kurs aufmerksam verfolgt haben: Bilden Sie eine Schleife für PRINT# und legen Sie die Wörter in DATA-Zeilen ab! Wer sehen will, ob er alles richtig gemacht hat, kann es mit dem Listing **print#** unserer Diskette vergleichen. Die erzeugte SEQ-Datei läßt sich mit **get#** wieder in den Computer holen, wenn Sie in Zeile 10 des Listings den Dateinamen in »sport« ändern.

Fühlen Sie sich inzwischen fit genug, gemeinsam mit uns ein Programmprojekt in Angriff zu nehmen? Gerade in Verbindung mit dem Speicher- und Laderoutinen der Floppy lassen sich komfortable und nützliche Datenbanken konstruieren. Dazu greifen wir auf unser bisher erworbenes Wissen über Basic 2.0 zurück. Das Programm soll **literatur-datei** heißen und uns ermöglichen, beliebige Bücher zu erfassen und auf Wunsch zu suchen. Die Datei, die damit erzeugt wird, benutzt zweidimensionale Felder und nennt sich »literatur«. Sie finden das entsprechende Programm auf der Diskette zum Sonderheft. Beachten Sie die Bedienungsanleitung und die Programmerläuterung im Kurzinfo-Kasten.

Der Grafikatlas

Faszinierend sind die grafischen Möglichkeiten des C64. Durch entsprechende Einstellung kann man den Textbildschirm mit seinen 25 Zeilen zu je 40 Spalten in eine hochauflösende Grafiklandkarte (Bitmap) verwandeln.

Was bedeutet hohe Auflösung? Der normale Textbildschirm bietet nach dem Einschalten des Computers 1000 markante Speicherstellen, die man entweder ein- oder ausschalten kann: 25 Zeilen x 40 Spalten = 1000 Plätze. In jedem Bildpunkt steht ein Zeichen (Charakter) des aktiven Zeichensatzes. Das können Buchstaben, Zahlen, Grafikzeichen von der Tastatur – oder Leerzeichen sein (dann ist der Bildschirm gelöscht!). Beachten Sie die Tabelle der Bildschirmcodes im Handbuch des C64. Eines haben alle Zeichen gemeinsam: Sie bestehen aus einer Matrix von 8 Byte mit jeweils 8 Bit, also 64 Bildpunkten (Pixel). Der Computer verfügt in einer Tabelle im Betriebssystem über 255 verschiedene Codes pro Zeichenmatrix. Je nachdem, welcher Wert angegeben wird, erscheint die betreffende 8 x 8-Pixel-Matrix an gewünschter Cursor-Position auf dem Bildschirm. Direkt beeinflussen lassen sich diese Pixel nur beim Ändern des Zeichensatzes – und da auch nur Byte für Byte pro Zeichen. Eine spezielle Speicherstelle hat einen Schalter, der den Anzeigemodus des Computers verändert: Bit 5 in $D011 (53265). Ist das Bit aktiviert (Wert 32), benutzt der Computer die »High Resolution« (Hires). Aus 1000 Bildpunkten sind nun 64mal soviel geworden:

8 Bit x 8 Byte x 1000 Bildstellen = 64000. Jetzt können Sie statt einer 8 x 8-Punkte-Matrix jedes einzelne Bit in diesem Feld beeinflussen: 1 = Bit eingeschaltet, 0 = Bit gelöscht. Als Berechnungsgrundlage wird die vom Textmodus bekannte Auflösung benützt: 40 Spalten zu jeweils 8 Bit ergeben 320 Pixel in horizontaler (x-) Richtung, 25 Zeilen pro 8 Byte registrieren 200 vertikale Bildpunkte (y-Richtung). Daran erkennt man, daß hochauflösende Grafik sehr speicheraufwendig ist: 64000 Bit benötigen 8000 Adressen im Computer (64000 : 8).

»Bitmapping« ist die meist verwendete Grafiktechnik beim Computer (nicht nur beim C64!), um detaillierte Bilder wiederzugeben (vergleichbar mit dem gerasterten Bild in einer Zeitung). Neben der Standard-Bitmap des C64 gibt's noch den Multicolormodus, der zwar vier verschiedene Farben bietet, dies aber mit einer geringeren Auflösung bezahlen muß: nur noch 160 Pixel in horizontaler Richtung. Wir wollen in unserem Basic-Kurs Wege zeigen, wie man diese Bitmaps in Basic erzeugt und aktiviert.

Dies ist zwar prinzipiell eine Aufgabe für Maschinensprache, denn Grafikbefehle gibt's im Basic 2.0 des C64 (im Gegensatz zu Basic 7.0 des C128) überhaupt nicht... Trotzdem: Es geht, mit den entsprechenden Rechenroutinen und dem Einsatz vieler Variablen! Schalten wir zunächst die Hires-Bitmap ein:

```
10 poke 53265,peek(53265)
   or 32
```

Bit 5 wurde durch die OR-Verknüpfung aktiviert. Nach dem Start unserer Basic-Zeile mit RUN erschient jedoch nur wirres Byte-Chaos auf dem Bildschirm. Drükken Sie nun die Tastenkombination <RUN/STOP RESTORE> oder geben Sie »blind« ein:

```
POKE 53265,
PEEK(53265)AND223
```

Damit ist der Originalzustand wiederhergestellt. Außerdem: Das Gewirr, das Sie gesehen haben, waren die ersten 8000 Byte des C64 (Speicherstellen 0 bis 7999), denn exakt dort liegt nun die Bitmap. Das bringt Probleme: Ändern wir dort irgendwelche Bits oder Bytes, dann verändern wir eventuell wichtige Speicheradressen der Zeropage (0 bis 255), des Speicherstapels (Stack, von 256 bis 511), der Betriebssystemvektoren ab 768 – und last not least unseres Basic-Programms, das ab Adresse 2049 beginnt. Diese Lage unserer Grafiklandkarte ist alles andere als günstig...wo soll man sie plazieren und wie läßt sie sich umstellen? Die Frage nach dem »Wo« ist schnell beantwortet: Innerhalb eines Bereichs von 16 KByte (= 16384 Byte), den der Videochip VIC-II überblicken kann. Da die ersten 8000 Byte aus den genannten Gründen nicht in Frage kommen, sind's eben die zweiten: ab Adresse 8192 bis 16191. Teilen wir die Anfangsadresse durch »1024« und POKEn das Ergebnis (8) in die Speicherstelle 53272:

```
20 poke 53272,peek(53272)
   or 8
```

Da der Beginn der Bitmap (8192) bei unseren weiteren Berechnungen noch eine Rolle spielen wird, speichern wir diese Zahl am besten in einer Variablen:

```
5 ba=8192
20 poke 53272,peek(53272)
or    ba/1024
```

Um uns des Byte-Gewirrs auf dem Bildschirm (es zeigt die aktuellen, zufälligen Inhalte der Adressen 8192 bis 16191) zu entledigen, muß dieser Bereich gelöscht werden (pro Byte eine »0« eintragen!):

```
30 for i=ba to ba+7999
40 poke i,0
50 next i
```

Nach RUN kann man beobachten, wie sich das Chaos auflöst und einem freien Bildschirm Platz macht. Ganz stimmt das nicht: Es bleiben noch immer einige undefinierbare Farbblöcke zurück: Sie haben mit der Hires-Bitmap nichts zu tun, sondern mit dem Farbspeicher für die Grafiklandkarte. Diese Farben kommen nun nicht mehr aus den Adressen 55296 bis 56295 (wie im Textmodus), sondern wurden vom Computer automatisch nach 1024 bis 2023 verlegt (im Textmodus ist das der Bildschirmspeicher). Da es aber nach wie vor nur 1000 Byte geblieben sind, muß man sich damit abfinden, daß sich trotz hochauflösender Grafik immer nur 8 x 8 Pixel verschiedenfarbig anzeigen lassen: Pro Pixel eine eigene Farbe, das würde nochmals 64000 Byte erfordern (die der C64 nicht hat).

Wie stellt man die Farbe ein? Bei der Standard-Bitmap gibt's nur zwei Farben: Zeichenfarbe (Vordergrund) und Hintergrund. Die

Rahmenfarbe bestimmt nach wie vor die Speicherstelle 53280 (wie schon aus dem Textmodus bekannt). Das Farb-Byte muß nun in zwei Hälften geteilt werden: Die oberen 4 Bit (deren Gesamtwert multipliziert man mit »16«) sind für die Vordergrundfarbe, die unteren 4 Bit (sie behalten ihren realen Wert) für den Hintergrund zuständig. Ein Beispiel: Sie möchten eine gelbe Hintergrundfarbe mit blauen Grafikpixeln. Gelb hat den Farbcode »7«, Blau die Nummer »6«. Die Rechenformel sieht dann so aus (C ist z.B. die Farbvariable):

```
55 c = 16 * 6 + 7
```

Um diese Farbwahl allen 1000 Speicherplätzen des Hires-Farb-RAM mitzuteilen, müssen wir wieder eine Schleife benutzen:

```
60 for i=1024 to 2023
70 poke i,c
80 next i
```

Schneller als beim Löschen des Hires-Bereichs füllt sich nun der Bildschirm mit gelber Farbe.

Um ein Pixel zu setzen oder zu löschen, muß der Computer natürlich wissen, wie er das richtige Bit im Speicher findet. Das bedeutet: Er muß das zu ändernde Zeichen, die Zeichenreihe und das entsprechende Bit in dieser Reihe identifizieren. Für den C64 existiert nämlich theoretisch noch immer der Textbildschirm, der mit 25 Zeilen 40 Spalten vollgeschrieben ist, nur läßt er im Hires-Modus zu, daß man den Speicherbereich anders interpretiert (**Abb.9**). Um die exakte Position eines Bildpunktes zu fixieren, gibt's bestimmte Rechenfunktionen, die man numerischen Variablen zuweist. Den Beginn des Hires-Bildschirms haben wir bereits mit BA = 8192 festgelegt. Die Position in der Bildschirmzeile errechnet sich so:

```
RO = INT(Y/8)
```

Das zugehörige Zeichen (Charakter) wird mit dieser Formel berechnet:

```
CH = INT(X/8)
```

Die Byte-Zeile innerhalb dieser Zeichenmatrix:

```
LI = Y AND 7
```

Das entsprechende Bit aus diesem Byte:

```
BI = 7-(X AND 7)
```

Alle Formeln muß man nun zusammensetzen. Die Byte-Zeile, in der das Pixel liegt, kann man jetzt eingrenzen:

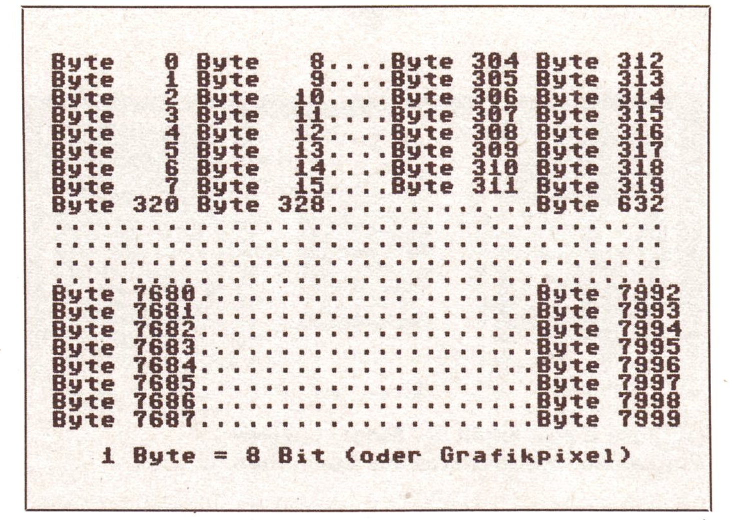
```
BY = BA + RO*320 + CH*8 +
LI
```

Endlich ist's soweit! Der gewünschte Bildpunkt erscheint mit dieser POKE-Anweisung auf der Grafik-Bitmap:

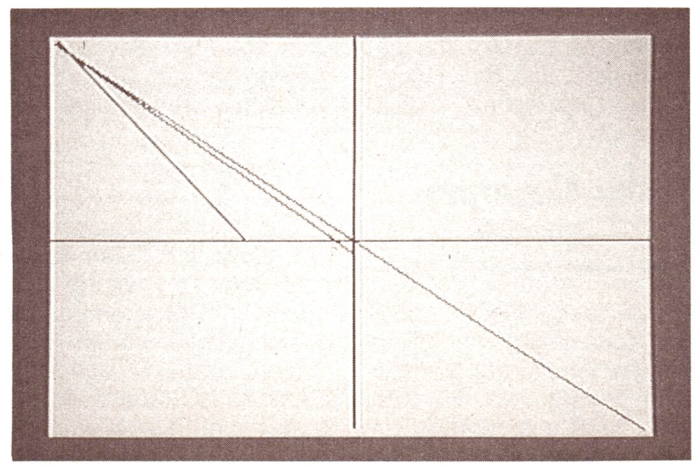
```
POKE BY, PEEK(BY) OR 2BI
```

Dazu gibt's fünf Demoprogramme unserer Diskette. **setpixel** zeigt, wie man Bitmaps einschaltet, das Farb-RAM initialisiert und nach Eingabe der x- und y-Koordinate einen Bildpunkt in der Hires-Grafik erzeugt. Da der Grafikbildschirm nur das erste Mal gelöscht wird, bleibt das eingeschaltete Pixel erhalten. Wenn Sie mit einem Tastendruck zurück zur nächsten Koordinateneingabe schalten, lassen sich so beliebige Grafiken entwerfen. Ein wenig mühsam ist es schon, da man Punkt für Punkt eintragen muß.

Das Programm **draw.bas** geht bereits einen Schritt weiter: Damit kann man ganze Linien in der Bitmap fabrizieren (**Abb.10**). Sie müssen lediglich Start- und Endkoordinaten der Linie angeben, Länge und Bildpunkte dazwischen berechnet das Programm automatisch und setzt an entsprechender Stelle ein Pixel. Dazu bildet es eine Schleife, deren Wert für die Laufvariable aus der Differenz zwischen Start und Ende gebildet wird. Da bei diesem Beispielprogramm vor jedem neuen Durchlauf die Grafiklandkarte nicht mehr gelöscht wird, kann man problemlos mehrere Linien eintragen und somit geometrische Gebilde erzeugen.



[9] Die Aufteilung der Grafik-Bytes im Hires-Modus



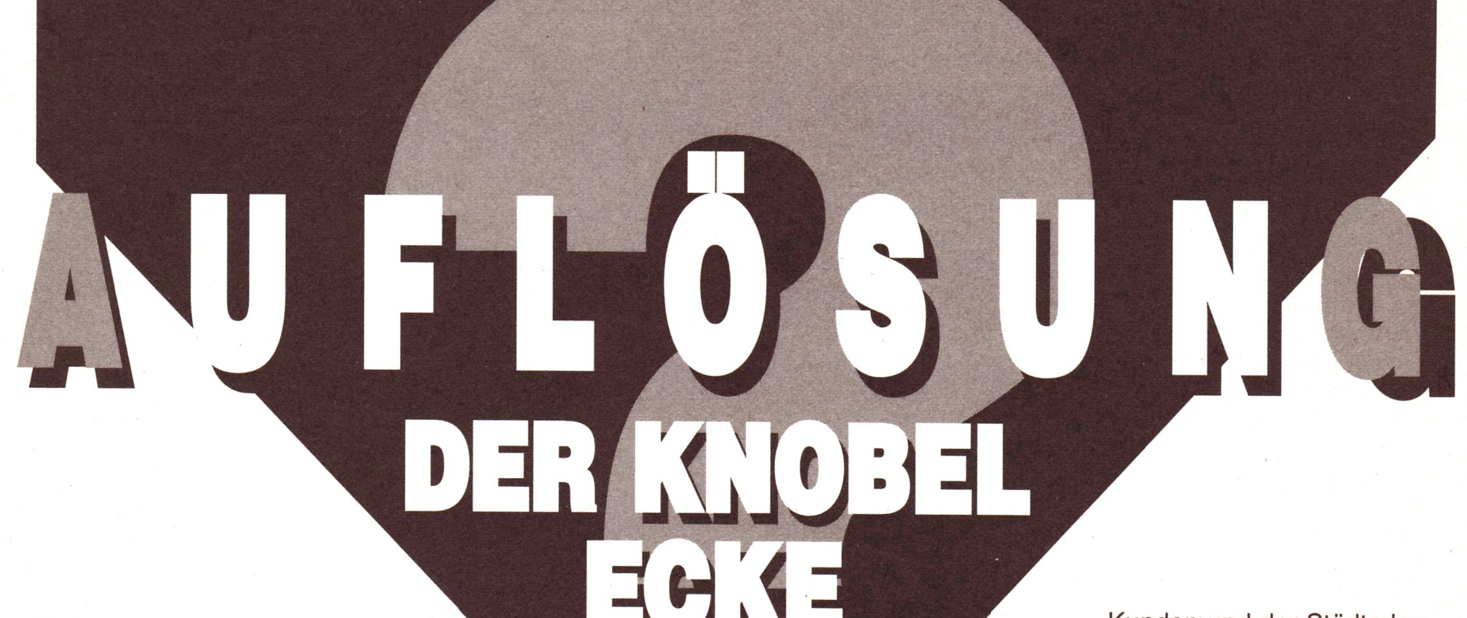
[10] Hires-Grafik und Basic 2.0: Viel Zeit vergeht...

box.bas macht mehr aus dem diesem Beispielprogramm: Umständliche Koordinateneingaben entfallen. Sie sind bereits als DATA-Zeilen im Programm abgelegt und werden durch den READ-Befehl wieder in den Variablenspeicher geholt. Man sieht Rechtecke unterschiedlicher Größe auf dem Bildschirm. Die Zeichengeschwindigkeit läßt allerdings zu wünschen übrig!

Was wären grafische Gebilde auf einer Hires-Bitmap ohne Kreise? **circle.bas** zeigt, daß sich auch dieser Grafikkörper mit Basic-Befehlen realisieren läßt. Wer glaubte, daß man mit dem Radius und der Ludolf'schen Zahl Pi (das ist die Konstante 3,14159265, die bei Kreisberechnungen eingesetzt wird) arbeiten muß, sieht sich eines Besseren belehrt: Der Programmierer benutzt zur Berechnung die SQR-Funktion des Basic 2.0, die eine Quadratwurzel aus einer beliebigen Zahl bildet. Durch geschickte Programmierung (s. Listing und Kurzinfo) wird so, je nach Wert der Koordinaten, auf dem Bildschirm ein Halbkreis erzeugt. Durch Vertauschen der Vorzeichen in der Rechenroutine zeichnet der Computer je einen Halbkreis auf dem unteren und oberen Bildschirm (zusammengefügt ergibt das den ganzen Kreis).

Alle Grafikelemente (Linie, Rechteck und Kreis) vereinigt das letzte Programmbeispiel zur Hires-Grafik: **fussballplatz**, das ein Sportstadion aus der Vogelperspektive zeigt.

Eines haben alle Basic-Beispiele zur hochauflösenden Grafik gemeinsam: Es vergeht viel Zeit, bis das gewünschte Ergebnis erscheint. Basic ist eine leicht erlernbare und einfache Computersprache, die aber – richtig eingesetzt – optimale Wirkung erzielt. (bl)

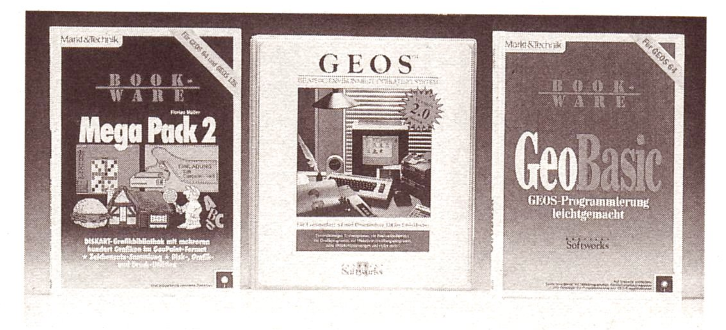


Erinnern Sie sich ans 64'er-Sonderheft 56? Die Programmieraufgabe in der Knobelecke war eine Herausforderung an alle Rätselfans! Hier ist das Siegerprogramm.

von Jürgen Weiland

Zu gewinnen gab's drei Leckerbissen: Geos 2.0 für den C64, Geobasic und Megapack 2 **(Abb.1)**. Folgende Aufgabe mußte durch ein Programm – Basic oder Assembler – gelöst werden: Die Spedition Bitberger in Bytehausen will ihre Kunden beliefern. **Abb.2** zeigt den Stadtplan mit allen neun Kunden und dem Standort des Spediteurs. Sie sollten ein Programm entwickeln, das den kürzesten Lieferweg berechnet: Nach dem Start mußte man die Koordinaten von Spedition und Kunden angeben. Wenn der LKW nach getaner Arbeit wieder in den Speditionshof zurückgekehrt war, mußte das Programm sowohl Rechenzeit als auch Anzahl der Straßen ausgeben. Großen Wert legten wir auf Benutzerkomfort. Das Programm mußte auf jeder Version des C64 lauffähig sein. Basic-Erweiterungen oder Hardwaremodule waren verboten!

Nachträglich beschleicht uns das bange Gefühl, daß die Aufgabenstellung zu schwer war: Auch bei dieser Knobelecke schaffte es wieder nur ein Leser, die geforderten Bedingungen in **allen** Punkten zu erfüllen: Jürgen Weiland **(s. Kasten)**. Ihm gehören jetzt alle drei Hauptpreise: Herzlichen Glückwunsch!



[1] Diese Preise erhält der Sieger unseres Programmierwettbewerbs aus Sonderheft 56

Das Siegerprogramm

»Beste Wege« finden Sie in der schnelleren, kompilierten Version auf der Diskette zum Sonderheft. Sie läßt sich wie ein Basic-Programm laden:

```
LOAD "BESTE WEGE",8
```

Gestartet wird das Programm mit RUN.

Die Berechnung des kürzesten Weges dauert ca. vier Minuten. 64 Straßen werden benutzt, die Kunden in dieser Reihenfolge besucht:

```
S,4,8,1,5,2,9,6,3,7,S
```

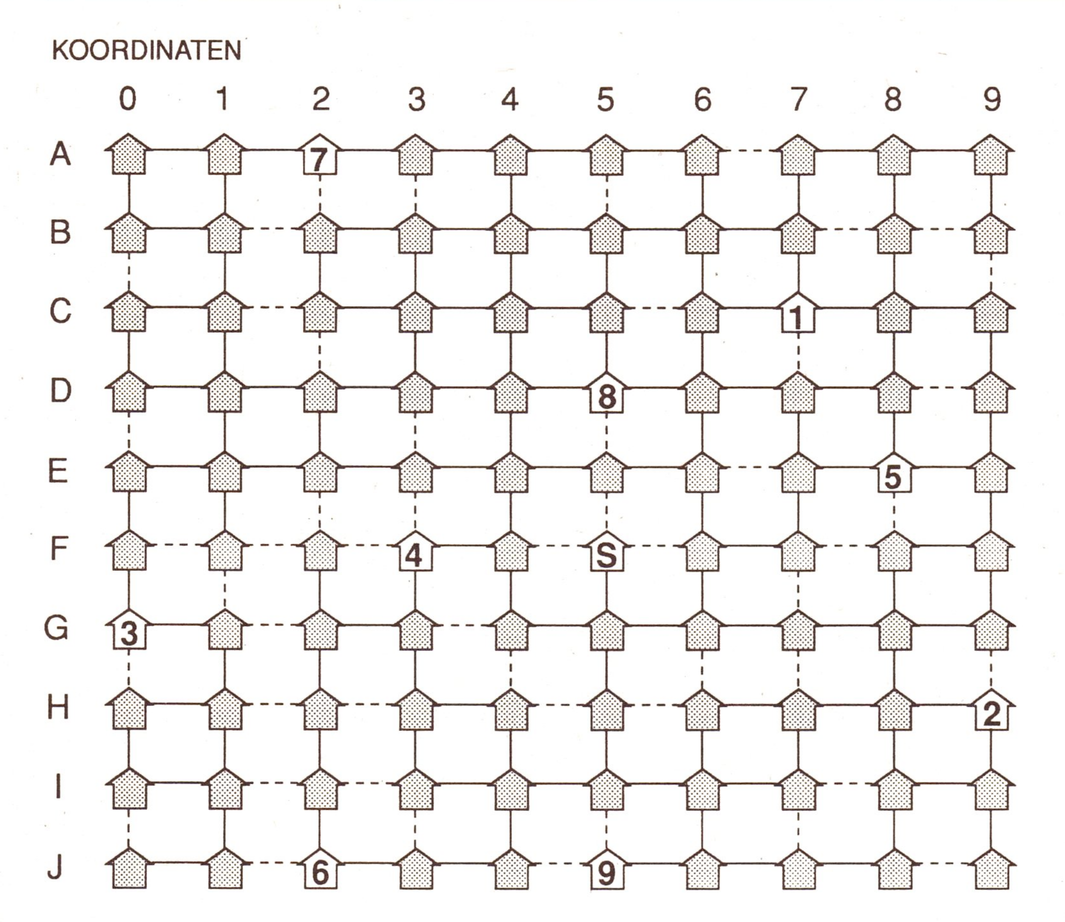
Maximal neun Kunden lassen sich mit »Beste Wege« verwalten. Man kann nach Wunsch wieder zur Spedition zurückkehren, die gestrichelten Straßen benutzen, mit dem integrierten Editor neue Stadtpläne entwerfen und die Lösungen zum Drucker schicken. Die Namen der Spedition, der Kunden und der Städte lassen sich beliebig umtaufen. Ebenso gibt's eine Einstellung, daß das Programm den kürzesten Weg zwischen nur zwei Kunden sucht. Eine grafische Spielerei rundet die Programmfunktionen ab: Innerhalb des Stadtplans kann man sogar Grünflächen pflanzen!

Der aktuelle Stadtplan befindet sich während des gesamten Programmablaufs auf dem Bildschirm. Alle Ein- und Ausgaben erscheinen entweder in der linken Bildschirmhälfte oder in separaten Windows. Die meisten Funktionen lassen sich mit der STOP-Taste abbrechen. »Beste Wege« benutzt einen geänderten Zeichensatz mit deutschen Umlauten, die über diese Tasten erreichbar sind:
ü = <@>, ä= <;>, ö= <:>, ß= < >. Für Großbuchstaben drücken Sie die SHIFT-Taste.

Im Hauptmenü sind Programmfunktionen enthalten, die mit der entsprechenden Zahlentaste aktiviert werden:

<1> Editor

Zunächst erscheint eine Übersicht aller Tasten, die in »Beste Wege« bestimmte Funktionen ausführen **(Tabelle)**. Erneuter Tastendruck verbannt sie wieder vom Bildschirm. Im weiteren Programmverlauf kann man dieses Hilfsmenü jederzeit per <F1>-Taste aufrufen.



[2] Die Aufgabe der Knobelecke aus dem 64'er-Sonderheft 56: Wer findet den kürzesten Lieferweg?

Mit <RETURN> lassen sich, vom Stadtplan positionsabhängig, Objekte setzen oder löschen. Auf Leerflächen zwischen den Häusern kann man Grünzeug plazieren. Wenn der Cursor direkt auf einem Haus liegt, verteilt der Computer aufsteigende Kundennummern. Erneuter Druck auf <RETURN> löscht sie wieder.

<SPACE> leitet die Berechnung ein: Damit können Sie im entsprechenden Eingabefenster den kürzesten Weg zwischen zwei Häusern suchen und auf dem Bildschirm ausgeben lassen. Nach Auswahl der beiden Lieferorte läßt sich mit der betreffenden Abfrage wählen, ob nur Straßen oder auch die Leerflächen dazwischen benutzt werden dürfen. Dies wirkt sich selbstverständlich auf die Rechengeschwindigkeit aus.

Die Namen der Stadt, Spedition und Kunden lassen sich nach Wunsch ändern. Wenn Sie die Tastenkombination <SHIFT N> drücken, bringt der Bildschirm ein Eingabefenster, in dem die bisherige Städtebezeichnung steht. Die müssen Sie jetzt überschreiben und <RETURN> drücken. Denken Sie dabei an die neue Tastenbelegung für die Umlaute!

Mit <S> kann man die Spedition an die aktuelle Position des Editor-Cursors setzen. Das entsprechende Zeichen besteht ebenfalls aus einem Häuserumriß, der aber im Gegensatz zu echten Häusern leer ist. Befand sich dort eine bereits definierte Kundennummer, wird sie allerdings gelöscht. War das Speditionszeichen bereits an anderer Stelle plaziert, verschwindet es von dort. Dies gilt auch für numerierte Häuser, wenn Sie diese per Zahlentaste einstellen (den entsprechenden Kundennamen dazu finden Sie in der linken Bildschirmhälfte).

Die Taste <CBM> ist der Ausgangspunkt für Diskettenfunktionen:

<CBM S> speichert alle aktuellen Daten (Stadtplan und Namen) als Datei, deren Bezeichnung beliebig sein darf (nicht länger als 16 Zeichen!). Geben Sie statt des Namens das <$>-Zeichen ein, bringt der Computer das Directory in einem speziellen Window. Mit dem Auswahl-Cursor läßt sich ein Dateiname bestimmen, der dann im oberen Eingabefenster erscheint. Nach <RETURN> erscheint die Frage, ob die gleichnamige Datei auf der Diskette mit den aktualisierten Daten überschrieben werden darf.

<2> Weg suchen

Mit dieser Funktion beweist das Programm seine Stärke: Es zeigt auf dem Stadtplan exakt, wie die Kunden angefahren werden. Voraussetzung ist allerdings, daß Sie vorher im Editormodus Kunden und Spedition definiert haben. Zunächst können Sie bestimmen, ob sich Leerstellen zwischen den Häusern befahren lassen und der LKW nach erledigter Kundentour wieder zur Spedition zurückkehren soll. Zusätzlich besteht eine Reihenfolge, in der die Kunden beliefert werden. Nach Abschluß der Eingaben (mit der Taste <RETURN>) beginnt die Arbeit des Programms. Durch einen ausgeklügelten Algorithmus hält sich die Rechenzeit in Grenzen. Wo der Computer die besten Lieferwege im Stadtplan findet, kennzeichnet er sie weiß. Während des Suchvorgangs läßt sich der Programmablauf mit <SHIFT> anhalten, durch die Tasten <CBM> und <CTRL> bremsen oder mit <RUN/STOP> abbrechen. Die Rechenzeit, die dann im angezeigten Auswertungsfenster erscheint, wird von den genannten Unterbrechungen nicht berührt.

Mit der Leertaste kommt der Stadtplan hinter dem Ausgabe-Window wieder zum Vorschein, <F7> wirft die Frage auf, ob die Rechenergebnisse auf Bildschirm oder Drucker ausgegeben werden sollen. Geschieht es auf dem Bild-

schirm, stoppt die Ausgabe nach jeweils 23 Zeilen und wartet auf einen Tastenimpuls, bevor es weitergeht. Mit <SHIFT> kann man die Ausgabe anhalten.

Druckt Umlaute

Wenn Sie die Wegfolge einzelner Häuser untereinander interessiert, wählen Sie die entsprechende Zahlentaste. <S> oder <0> ist auch hier das Kennzeichen für die Spedition. Die Rückkehr ins Hauptmenü funktioniert nur mit der Taste <F7>.

<**3**> **Drucker**

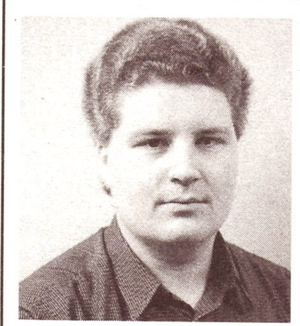
Funktioniert er oder nicht? Diese Kardinalfrage stellt sich mit dieser komfortablen Menüfunktion nicht: Die entsprechenden Druckerparameter kann man bequem anpassen (Voreinstellungen in Klammern):
- Sekundäradresse (07),
- ESC-Codes für den ASCII-Modus (1B 4F 1B 6C 05),
- Druckerinitialisierung (1B 55 01),
- neue Seite (0C),
- maximale Zeilenanzahl (38).

Da das Programm keine Hires-Grafik benutzt, gibt es mit Epson- und Commodore-kompatiblen Druckern keine Schwierigkeiten. Umlaute werden automatisch in Doppelzeichen umgewandelt (z.B. »ae« statt »ä«). Die Geräteadresse kann auf »4« oder »5« definiert sein: Das Programm erkennt es automatisch.

<**4**> **Ende**

Nach einer Sicherheitsabfrage können Sie das Programm verlassen. Der C64 führt allerdings keinen Reset aus: »Beste Wege« läßt sich nach RUN erneut starten.

Jürgen Weiland



wurde 1966 in Marl (Westfalen) geboren. Am Gymnasium im Loekamp machte er 1985 das Abitur. Nach der Bundeswehrzeit begann er zunächst, in Aachen Elektrotechnik zu studieren. 1988 wechselte er auf der Universität Koblenz den Studienzweig: Informatik.

Durch seinen Schwager (ebenfalls Informatikstudent) kam er in den 70er Jahren das erste Mal mit Computern in Berührung. 1983 kaufte er sich einen TI-99/4A. Mitte 1984 sattelte er auf den C64 um, der durch ständige Erweiterungen und Umbauten nicht mehr viel mit dem Original gemeinsam hat. Obwohl Jürgen durchs Studium wesentlich leistungsfähigere Computer kennengelernt hat, bleibt er seinem C64 nach wie vor treu. Für ihn reicht er – zumindest zu Hause – völlig aus.

Kurzinfo: Beste Wege

Programmart: Anwendung
Laden: LOAD "BESTE WEGE",8
Starten: nach dem Laden RUN eingeben
Steuerung: Tastatur
Besonderheiten: Editor für Stadtpläne, Druckerprotokoll
Benötigte Blocks: 202
Programmautor: Jürgen Weiland

Auf der Suche

Wie funktioniert das Programm? Ausgehend von allen Kunden und dem Standort der Spedition ermittelt »Beste Wege« die kürzesten Verbindungen. Dabei werden alle möglichen Richtungen getestet: Zuerst die, in der sich rein logisch das Ziel befindet (aktuelle Positionen des Start- und Zielhauses). Das ist die erste Heuristik, die das Programm verwendet.

Über die Bedeutung dieses griechischen Wortes belehrt uns das Lexikon: die Lehre von Methoden zur Auffindung neuer wissenschaftlicher Erkenntnisse.

Weitere heuristische Prinzipien, die im Programm vorkommen:
- Hat es eine Lösung gefunden, bricht es die Suche bereits ab, wenn sich ein längerer Weg ergeben würde.
- Überschreitet die Wegstrecke einen gewissen Erfahrungswert, wird die Suche an dieser Stelle abgebrochen.
- Im Normalfall sucht »Beste Wege« ab der niedrigeren Nummer zur nächsthöheren. Findet das Programm aber nach einer bestimmten Zeit keine Lösung, sucht es in umgekehrter Reihenfolge. Oft wird damit die Lösung schneller gefunden. Es hängt nicht zuletzt von der Richtung der Häuser im Stadtplan zueinander ab.

Führen die erwähnten Heuristiken nicht zum Erfolg, schaltet das Programm aufs Backtracking-Verfahren um: Es sucht so lange, bis es die Lösung gefunden hat. Unmögliche Aufgaben (z.B. keine Verbindung zwischen zwei Kunden) quittiert das Programm mit einer Fehlermeldung und bricht die Suche ab. Aus allen gefunden Einzellösungen ermittelt der Computer den kürzesten Weg und entscheidet sich dafür (Greedy-Verfahren).

Die Lösung

Wer die Auflösung unserer Knobelecke im 64'er-Sonderheft 56 mit dem Programm nachvollziehen will, muß:
- im Hauptmenü die Taste <1> (Editor) drücken,
- die Tastenkombination <CBM L> aktivieren, und
- als Dateiname »bytehausen« eingeben oder das File aus dem Directory laden.

Wenn der Stadtplan im Computer ist und auf dem Bildschirm erscheint, erkennen Sie, daß er mit unserer Aufgabenstellung identisch ist. Mit der Taste <**Pfeil nach links**> kehren Sie zurück ins Hauptmenü.

Drücken Sie jetzt die Taste <2> (Weg suchen). Beantworten Sie die folgenden Fragen:

Leerstellen befahrbar: <N>,

Rückkehr zur Spedition: <J>,

Reihenfolge vorgeben: <N> (schließlich soll das Lösungsprogramm den optimalen Lieferweg selbst herausfinden!).

Die Suche beginnt, nach 4,28 min ist sie beendet: Die Lösung erscheint.

Die Knobelaufgabe hat dem Autor offensichtlich Spaß gemacht, denn er hat außer »bytehausen« noch fünf weitere Stadtpläne mit unterschiedlicher Schwierigkeit entworfen:

```
bytehausen-opti
kleinstadt I
kleinstadt II
kleinstadt III
schlingenhausen
```

Laden Sie diese Dateien im Editor und lassen Sie das Programm die besten Wege suchen. Selbstverständlich hindert Sie niemand daran, völlig neue Stadtpläne zu entwerfen.

(N. Heusler/bl)

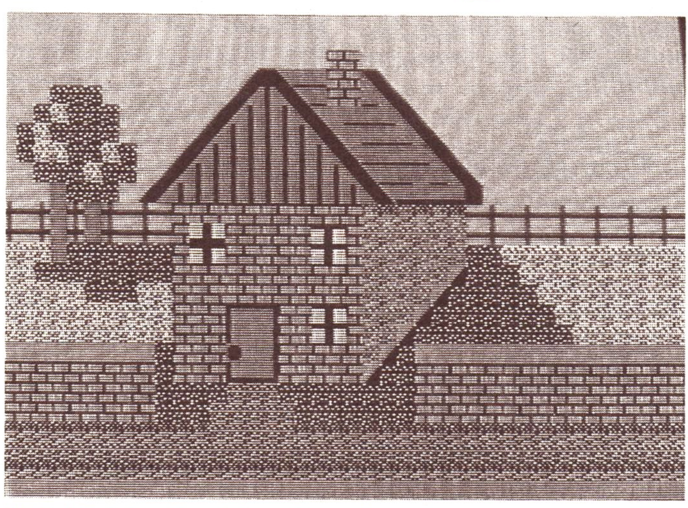
Editiermodus in »Beste Wege«

Taste	Funktion
<**F1**>	Befehlsliste anzeigen
<**CRSR**>	Cursor im Stadtplan bewegen
<**SHIFT N**>	Städtenamen ändern
<**SHIFT S**>	Speditionsbezeichnung ändern
<**RETURN**>	Objekte im Stadtplan setzen/löschen
<**S**>	Position der Spedition
<**0**>	s. oben
<**1 - 9**>	Kundennummern vergeben
<**N**>	Name des Kunden ändern, auf dessen Haus der Cursor steht
<**SPACE**>	kürzesten Weg zwischen zwei Häusern suchen
<**←**>	Hauptmenü
<**CBM S**>	Parameter zum aktuellen Stadtplan speichern
<**CBM L**>	Stadtplan laden (USR-Files)
<**CBM D**>	Stadtplan auf Diskette löschen

Kniffe, Tips & Tricks in Basic

Was das Handbuch verschweigt...

Sie sind das Salz in der Suppe: Die Tricks der Programmierprofis verleihen Basic-Programmen den letzten Schliff und führen natürlich auch Anfänger schneller zum Ziel.



[1] Bei Spielen bestehen die meisten Levelbilder aus einem geänderten Zeichensatz

Von Grafikbanken und -punkten

Wir haben beschrieben, wie man hochauflösende Grafik aktiviert. Das galt allerdings nur unter der Voraussetzung, daß Ihr Basic-Programm nicht länger als ca. 6000 Byte ist: Sonst kommt es mit der Hires-Bitmap in Konflikt. Eng begrenzt ist der Speicherplatz der VIC-Bank 1: Zeropage, Basic-Programm, Variablen und Bitmap, zusammengepfercht auf 16 KByte. Auf der Suche nach einem ungestörten Plätzchen stößt man auf die ideale VIC-Bank 2: von 16384 bis 32767. Man setzt die Grafiklandkarte in den zweiten Teil der Bank (ab 24576) und verschiebt den Bildschirm ebenfalls (1024 Speicherstellen darunter, ab 23552). Ihr Vorhaben müssen Sie dem Computer mitteilen:

```
POKE 56576, PEEK(56576)
AND 252) OR 2
POKE 56578, PEEK(56578) OR3
```

Diese beiden Adressen im Speicherbaustein CIA II sind dafür zuständig. Zwei weitere POKEs sind notwendig:

```
POKE 53272, 120: POKE 648,
92
```

Wenn Sie Bit 3 der Adresse 53272 auf 1 setzen, weiß der VIC-Chip, daß sich die Bitmap in der zweiten Hälfte befindet. Die Bits 4 bis 7 teilen dem Computer mit, wohin der Bildschirmspeicher versetzt wurde. Außerdem muß das Betriebssystem über Speicherstelle 648 informiert werden, ab welchem Speicherplatz die Bildschirmdaten liegen (23552 : 256 = 92). Zwar ist die Bitmap jetzt ziemlich weit oben im Speicher (Vorteil: Basic-Programme können nun erheblich länger sein, mehr als 20000 Byte stehen zur Verfügung), aber dennoch im Basic-RAM. Die Grafik könnte durch Variablen überschrieben werden. Dem ist abzuhelfen: Gaukeln Sie dem C64 vor, sein Speicher sei kleiner:

```
POKE 51,0: POKE 52,92
POKE 55,0: POKE 56,92
```

Das sind die Zeiger auf das Ende der Variablenfelder und des gesamten Variablenspeichers: Nun reicht er bis maximal 23552 (92 X 256).

Bekannt ist inzwischen auch, wie die Grafikpixel auf dem Bildschirm berechnet und plaziert werden. So sehr häufiger Variableneinsatz in Basic-Programmen zu empfehlen ist: Sie bremsen den Ablauf der Rechen- und Pixel-Setzroutine erheblich. Einfacher und schneller geht's, den gesamten Vorgang in drei Basic-Zeilen zu definieren:

```
BA = 24576
BY = (X AND 504)+40*(Y AND
248)+(Y AND 7):BI=7-(X AND-
7)
POKE BA+BY, PEEK (BA+BY)
OR (2↑BI)
```

Wir haben die neue Routine mit der aus dem Basic-Kurs verglichen: Zum Ziehen einer durchgehenden Linie benötigt sie 6 s weniger!

Länger dauert dagegen das Löschen der Bitmap durch Basic-Anweisungen. Hier hilft wirklich nur Maschinensprache. Die entsprechende Routine finden Sie im Grafikbeispiel **speedy hires** auf unserer Diskette, das die neue Punktesetzroutine verwendet. Die Zeilen 1000 bis 1030 beinhalten die Maschinensprache-Data mit Einleseschleife. (A. Locker/bl)

Neuer Zeichensatz gefällig?

So einfach, wie sich's anhört, ist es leider nicht: Der C64 holt seine Bildschirmzeichen (Characters) aus dem Zeichensatz-ROM ab $D000 (53248 bis 57343). ROM (Read Only Memory) ist ein Speicherbereich, den man nur lesen – aber nicht mit POKE beschreiben kann. Wer seinen Computer z.B. mit deutschen Umlauten, Sonderzeichen oder Mauersteinen für den Level eines Spiels ausstatten will, muß den Zeichensatzbereich an eine passende Stelle ins RAM kopieren (in VIC-Bank 1 liegt der ideale Bereich ab Adresse $3000, 12288 dez.) und die Zeiger in den entsprechenden Speicherstellen darauf richten. Diese Basic-Schleife macht's:

```
FOR I=0 TO 4095
POKE 12288+I,PEEK(53248+I)
NEXT
POKE 53272,PEEK(53272) OR
12
```

Gehen Sie ruhig inzwischen Kaffee trinken, denn es dauert leicht eine halbe Minute. Um so größer ist die Enttäuschung, wenn Sie wiederkommen. Lediglich die Bytes des VIC-, des SID- und der CIA-Chips wurden ins RAM übertragen (ganz zu schweigen vom Farb-RAM ab Adresse 55296), also der Ein- und Ausgabebereich (I/0-Register). Dieser Platz ist im C64 gleich dreimal belegt:

- I/0-Bereich,
- Zeichensatz und
- freies RAM.

Welche Konfiguration aktiv ist, bestimmt Adresse 1. Der Normalinhalt dieser Speicherstelle nach dem Einschalten des Computers ist »55« oder %0011011 als Binärzahl. Tabelle 1 macht die Funktionen der Bit-Belegungen deutlich (nur die ersten drei sind relevant, »1« = Bit ein-, »0« = Bit ausgeschaltet).

Es muß also Bit 2 (Wertigkeit: 4) gelöscht werden, um das Zeichensatz-ROM zu erwischen: POKE 1,51. Und schon tritt das nächste Problem auf: Durch diesen POKE wurde der I/0-Bereich abgeschaltet – der Compu-

ter gehorcht keiner Eingabe mehr! Schuld daran ist der Interrupt, den der C64 alle 1/60stel Sekunden ausführt (Tastatur abfragen, Bildschirm erneuern usw.). Wenn Sie diese Unterbrechungen auch noch abschalten (Speicherstelle 56334 auf »0« setzen), dann klappt's:

```
10 poke 56334,0: poke 1,51
20 for i=0 to 4095
30 poke 12288+i,peek(53248
+i)
40 nexti
50 poke 1,55: poke 56334,1
60 poke 53272,peek(53272)o-
r12
```

Zeile 60 bedarf einer Erläuterung: Die unteren 4 Bits der Adresse 53272 enhalten den Zeigerwert für den Beginn des Zeichensatzbereichs, den der Computer verwenden soll (man muß die Zahl mit »1024« multiplizieren: 12288). Jetzt können Sie die Byte im RAM nach Belieben ändern und ummodeln.

Auch dieses Verfahren ist nicht gerade ideal: Abgesehen von der Übertragungsdauer (30 s) bleibt Ihnen auch hier nicht viel Platz für Ihr Basic-Programm: knappe 12000 Byte. Wie im Beispielprogramm **speedy hires** ist zu empfehlen, die VIC-Bank 2 einzuschalten und z.B. dort einen fertig geänderten Zeichensatz an die relative Adresse 12288 (erhöht um »16384«, also »28672) zu laden (POKE-Wert für 53272: 188). Der Bildschirmspeicher muß ebenfalls relativ verschoben werden (am besten 1000 Byte darunter): 27648. Diesen Wert, geteilt durch »256«, müssen Sie zusätzlich in die Adresse 648 POKEn: 108 (Normalinhalt: 4). Damit stehen Ihnen immerhin 25600 freie Basic-Byte zur Verfügung.

Laden und starten Sie unser Programmbeispiel **charlader**. Sie haben die Wahl zwischen zwei fertigen Zeichensätzen- **amiga.char** und **game.char**, die jeweils 4096 Byte lang sind und 17 Blocks auf der Diskette umfassen. Geben Sie nach dem Start den gewünschten Zeichensatznamen ein. **game.char** bietet außer deutschen Umlauten auch einige geänderte Zeichen, die sich z.B. gut eignen, Levelbilder für Spiele zu entwerfen (**Abb.1**). Die neuen Zeichen finden Sie auf folgenden Tasten:
<@> = Ü, <[> = Ö, <]> = Ä <Pfeil hoch> = ß. <->, <>, <#>, <'> und <SHIFT A> sind Mauersegmente. <SHIFT S> und <SHIFT D> erzeugen Füllflächen. (bl)

Bremsfallschirm

Auch wenn er oft als zu langsam verwünscht wird: Bei der Bildschirmausgabe z.B. von Listings ist der C64 wieselflink – der Basic-Text huscht über den Bildschirm, bevor man ihn richtig lesen kann. Zwar gibt's mit <CTRL> eine Bremstaste, aber das Utility **bildsch. langsam** verlangsamt jede Ausgabe auf dem Bildschirm um ein Vielfaches. Nach dem Laden und Start mit RUN schreibt es den Maschinencode zunächst in den Bildschirmspeicher und verschiebt sich nach 53228. Alle Betriebssystemvektoren, die für die Bildschirmausgabe zuständig sind, werden nun auf diese Adresse gerichtet. Das Utility macht nichts anderes, als zwischen jedem Interrupt zwei Verzögerungsschleifen zu erzeugen. Mit <RUN/STOP RESTORE> oder

```
POKE 806,202: POKE 807,241
```

kann man die Erweiterung abschalten.
die Ausgabegeschwindigkeit läßt sich mit »POKE 53232, Zahl (0 bis 255)« regulieren. Voreingestellt ist »1«. (Dominik Irion/bl)

Bildschirm- und ASCII-Codes

Sie haben's sicher schon bemerkt: der Code für die Zeichen auf dem Bildschirm und die ASCII-Werte (CHR$) stimmen nicht immer überein. Überprüfen Sie folgendes Beispiel: Ein »A« läßt sich per Bildschirmcode so erzeugen:

```
POKE 1024,1
```

Wollen Sie das »A« als ASCII-Code erscheinen lassen, müssen Sie die Zahl um »64« erhöhen:

```
PRINT CHR$(65)
```

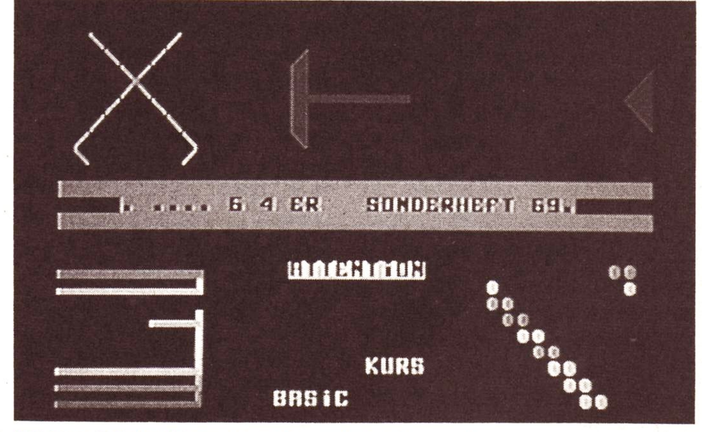
Die Basic-Routine **bs/ascii-code** nimmt Ihnen die Rechenarbeit ab. Nach dem Start mit RUN müssen Sie den gewünschten Bildschirmcode eingeben. Umgehend erhalten Sie den ASCII-Wert auf dem Bildschirm. Man muß allerdings beachten, daß jeder Bildschirmcode über »128« (inverse Zeichen) ebenfalls nur den ASCII-Code fürs normale Zeichen ergibt.
(Knut Smoczyk/bl)

C64 mit Rolladen

Man kennt sie von vielen Benutzeroberflächen für Computer (z.B Geos, die Workbench des Amiga usw.) Gemeint sind die komfortablen Pull-down-Menüs, die in der obersten Bildschirmzeile aktiviert werden. Reverse Cursor-Balken lassen sich per Tastatur, Maus oder Joystick auf den gewünschten Menüpunkt bewegen, der dann aufgerufen wird. Unsere Maschinenspracheroutine **pull-down** bietet allen Basic-Programmierern die Möglichkeit, solche Menüs in eigenen Programmen zu verwenden: Man muß lediglich die Assembler-Routine zu Programmbeginn laden (wie Zeile 90 unseres Beispielprogramms **pulldown-menue**). Die Namen der Menüpunkte sind als Stringvariablen definiert, die sich jederzeit nach Belieben ändern lassen. Zur Wahl der Menüpunkte dienen die Cursor-Tasten, mit <RETURN> ruft man die Funktion auf. Halten Sie sich bei eigenen Programmentwicklungen möglichst exakt an die Zeilennumerierung des Demoprogramms – davon ist abhängig, ob die richtige Programmreaktion aufgerufen wird (die entsprechenden Unterprogramme können ab Zeile 370 stehen).
(Said Baloui/bl)

Es geht abwärts

Das Ein-Zeiler-Programm **scroll n. unten** greift die bereits erwähnte Idee auf: Schreibt man eine Zeile auf den Bildschirm, die mehr als 40 Zeichenspalten umfaßt, scrollen alle darunterstehenden Zeilen um eine nach unten. Das Programm erzeugt den Scroll-Vorgang künstlich: Es benutzt diverse Cursor-Steuerzeichen und die Speicherstelle 218 (aktuelle Bildschirmzeile).
(Thomas Dreier/bl)



[2] Der Bildschirm scrollt, wohin Sie wollen ...

Zahlenumrechnung

Wer kennt auf Anhieb die entsprechende Hexzahl, die zu einem dezimalen Wert gehört? Noch mehr Kopfzerbrechen bereitet die Umwandlung ins binäre Zahlensystem (Dualzahlen). Unser Basic-Utility **zahlenwandler**

erledigt das kurz und bündig.

Nach dem Programmstart bietet ein Menü folgende Möglichkeiten, die mit den entsprechenden Tasten aufzurufen sind:
- Hexadezimal als Dezimalzahl,
- Dezimalwert als Hexzahl,
- Dezimal- als Dualzahl,
- Binärwert in dezimaler Darstellung.

Die Taste <E> beendet das Programm. Beachten Sie, daß bei der Eingabe für Dualzahlen (sie können nur aus den Ziffern »1« oder »0« bestehen) eine Eingabelänge von acht Stellen vorgeschrieben ist, z.B. muß die Zahl 15 so eingetippt werden: 00001111 (nur vier Einser ohne Nullen genügen nicht!). Die umgewandelte Zahl erscheint revers unter der eingegebenen Zahl. Im Menü können Sie nun die nächste Zahlendarstellung auswählen. Der Vorteil dieses Programms liegt darin, daß man sich bei der Hex- oder Dezimalzahl nicht auf die maximalen Werte des 8-Bit-Computers (0 bis 65535) beschränken muß: Lediglich bei den Dualzahlen lassen sich nur Werte zwischen 0 und 255 anzeigen.

(Martin und Hartmut Sprave/bl)

Automatische Zeilennumerierung

Dieses außerordentlich nützliche Utility kennen die meisten nur als Basic-Erweiterung in Maschinensprache: die automatische Vorgabe der Basic-Zeilennummern. Mit einem Trick geht's auch in Basic 2.0 – und es erfüllt denselben Zweck.

Tabelle 1. Die Bits der Speicherstelle 1

Bit.Nr.	Zustand	Funktion
0	1	Basic-ROM (40960 bis 49151)aktiv
	0	Basic-RAM eingeschaltet
1	1	Kernel-ROM (57344 bis 65535) ein
	0	RAM unter Kernel ein
2	1	I/O-Register (53248 bis 57343) ein
	0	Zeichensatz-ROM ein

Die Bits 3 bis 5 kümmern sich um den Datasettenbetrieb, Nr. 6 und 7 werden nicht benutzt (immer »0«)

Tabelle 2. Die Parameter von »Rolling«

Syntax-Vorschrift: SYS 50550,r,za,ze,s,l

r	Richtung	L,R,H,T
za	Zeilenanfang	1 bis 25
ze	Zeilenende	1 bis 25, za < ze
s	Spalte	1 bis 40
l	Zeilenlänge	1 bis 40, s + l < 41

Wichtig: Die Routine **auto.bas** besitzt sehr hohe Zeilennummern (63990 und 63991) und gibt eine Schrittweite von »10« vor. Diese Zahl läßt sich in Zeile 63991 ändern (A+10). Sind Sie mit der Eingabe Ihres Basic-Programms fertig, müssen die beiden Zeilen der Auto-Routine wieder gelöscht werden.

(Herbert Kunz/bl)

Eingabehilfe für Einzeiler

Dieses Programm **88 zeichen** macht Unmögliches möglich: Es lassen sich Basic-Zeilen bis zu maximal Länge von 88 Zeichen eingeben!

Nach RUN wird der Bildschirm gelöscht, der Cursor erscheint in der dritten Bildschirmzeile. Darunter (Zeile 4, Spalte 8) findet man den <Pfeil hoch>, der signalisiert: bis hierher und nicht weiter! Geben Sie nun eine überlange Basic-Zeile bis exakt zu dieser Position ein. Mit <RETURN> wird sie übernommen. Achtung: Vermeiden Sie beim Ausprobieren die Nummer 1 als Zeilennummer (die benützt das Miniprogramm selbst!). Wie funktioniert's? Die wichtige Fortsetzungszeile mit acht Speicherstellen erzeugt die Betriebssystemroutine ab Adresse 59749. Der C 64 benutzt sie ebenfalls, wenn 40 Zeichen in eine Bildschirmspalte eingetragen wurden: Scrollen nach unten, nächste Zeile löschen. SYS 42112 beendet ein Basic-Programm ohne READY-Meldung. Die entsprechenden Betriebssystemvektoren stehen in den Adresse 770/771.

(Jörg Peschel/bl)

Als die Bilder laufen lernten

Mit dem Programm **rolling** kann man Blockgrafik oder Text in Bildschirmbereichen scrollen lassen, die Sie frei wählen dürfen. Und das in alle vier Richtungen! Das ermöglichst allerdings erst das Maschinenspracheprogramm **roll.obj**, das vom Basic-Demo nachgeladen wird, ebenso wie der geänderte Zeichensatz **zs**. Hinweis: Es ist derselbe wie **game.char**, allerdings mit anderer Startadresse: $3000 (12288). Damit können Sie innerhalb der PRINT-Zeilen des Demoprogramms mit den geänderten Zeichen (Mauersegmente, Füllmuster usw.) experimentieren. Zur Initialisierung braucht die Routine einige Parameterangaben:

SYS 50550,R,ZA,ZE,S,L.

Es dürfen auch Variable sein. Was sie bedeuten, zeigt **Tabelle 2**. Achten Sie darauf, daß der Zeilenanfang nie größer als das Zeilenende ist. Außerdem darf die Summe aus Spaltenzahl und Zeilenlänge den Wert »41« nicht übersteigen. Unreelle Zahlen fängt das Programm mit einer Fehlermeldung ab. Experimentieren Sie mit dem Beispielprogramm **rolling (Abb.2)**. Die SYS-Aufrufe der einzelnen Bildschirmbereiche stehen in den Zeilen 50 bis 90.

(J. Effenberg/bl)

Superschnelles Directory in Basic

Es muß nicht immer Maschinensprache sein: Mit dem Basic-Programm **dir.bas** kommt das Disketteninhaltsverzeichnis ebenso schnell auf den Bildschirm wie mit unzähligen, vergleichbaren Assembler-Routinen. Das ist der Trick: Die Tastatur wird als Standard-Eingabegerät für die geöffnete Diskettendatei angegeben: Sekundäradresse 0. Jetzt kommen die Zeichen nicht mehr über das langweilige GET#, sondern durch den normalen GET-Befehl, der gewöhnlich Zeichen von der Tastatur einliest.

(Olaf Kummer/bl)

Restore X in Basic

Das Utility **goto x** auf der Diskette zum Sonderheft besitzt auch die Funktion »Restore X« (X = beliebige Zeilennummer oder numerische Variable). Nicht ganz so komfortabel, überdies arbeitsaufwendiger, aber nicht weniger effektiv läßt sich diese Aufgabe auch in Basic erfüllen. Dazu brauchen wir einige Adressen aus der Zeropage: 65/66 (Speicherzellen des aktuellen DATA-Elements) und 122/123 (Basic-Zeiger auf den nächsten Befehl innerhalb der Unterroutine). Pro Lesevorgang für die DATA-Zeilen benötigen Sie ein separates Unterprogramm zum Einlesen der Daten. POKEt man die Werte von 122/123 in die Speicherstelle 65/66, macht das Programm bei dem Wert weiter, der auf die Basic-Zeile mit den POKEs folgt: beim ersten DATA-Element im Unterprogramm (aber nicht im Gesamtprogramm, darauf kommt's an!). Nach dem Rücksprung aus der DATA-Unterroutine muß man auch die Zeiger im Hauptprogramm wieder regulieren. Unser Beispiellisting **restore.bas** ruft drei solcher Daten-Unterprogramme in zufälliger Rei-

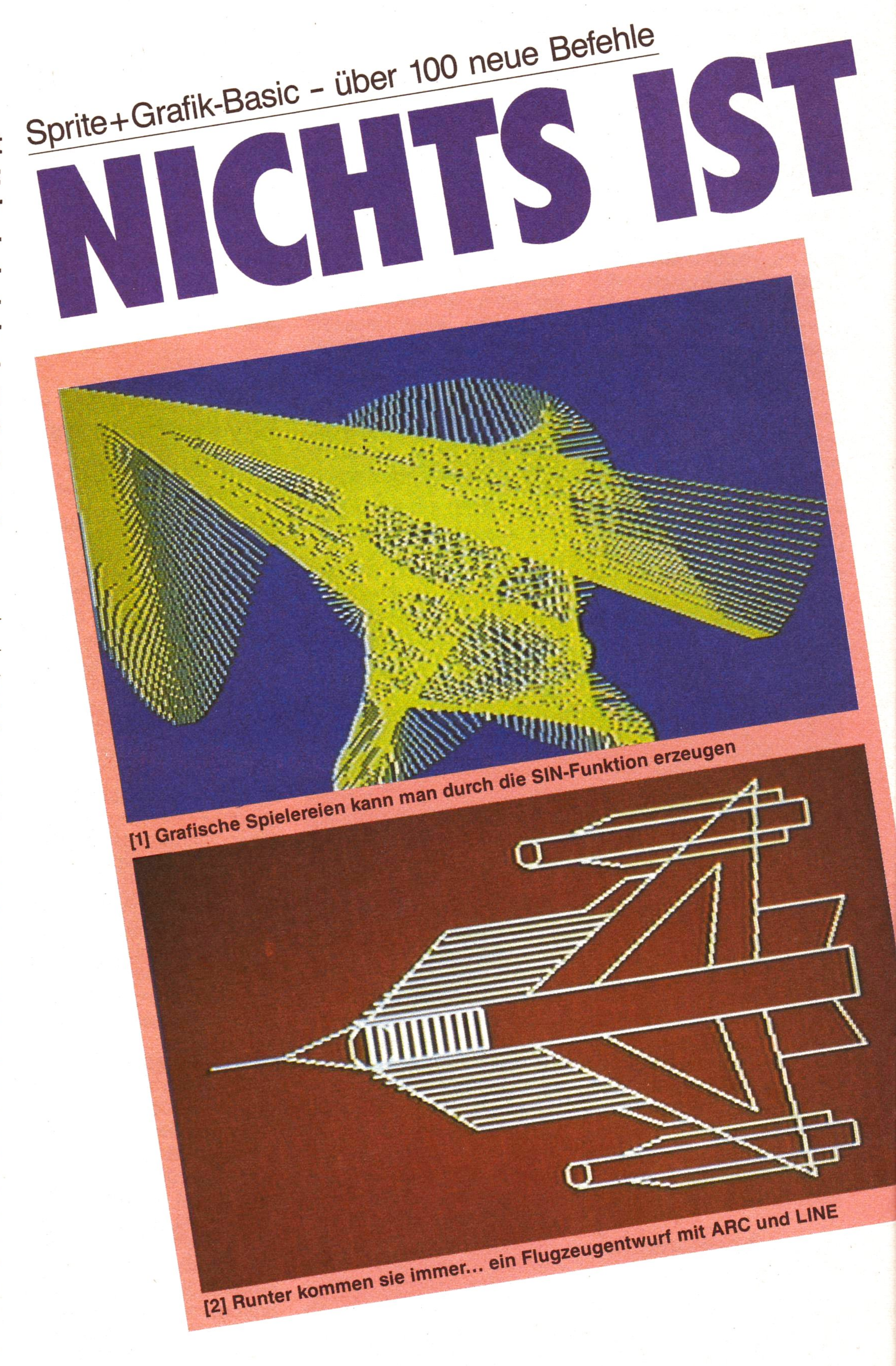
Fortsetzung Seite 50

Sprite+Grafik-Basic – über 100 neue Befehle

NICHTS IST

Komfortabler geht es fast nicht mehr: Hires, Multicolor und natürlich Sprites sind nur ein Vorzug dieser Super-Erweiterung. Aber auch die Blockgrafik kommt nicht zu kurz. Und damit die Programme nicht zu lang werden, gibt's jede Menge strukturierender Befehle.

von Jens Schwarz



[1] Grafische Spielereien kann man durch die SIN-Funktion erzeugen

[2] Runter kommen sie immer... ein Flugzeugentwurf mit ARC und LINE

Sie sind überall bekannt, die kleinen auf dem Bildschirm umherhuschenden Wesen, Sprites genannt. Sie hüpfen über Treppen, verfolgt von gierigen Monstern, überwinden Schluchten und sind trotz grausamster Tode bei jedem Spielbeginn wieder lebendig und tatenbereit. Wen juckt es da nicht in den Fingern, eigenen Kreaturen Leben einzuhauchen? Leider ist dies mit dem normalen Betriebssystem des C64 nicht ganz so einfach, wie es später im fertigen Werk aussieht. Eine tiefe Kenntnis des Betriebssystems und seiner Hardwarekomponenten ist Grundlage, um im normalen Basic auch nur an die Entstehung eines Sprites zu denken. Was tun? Greifen Sie zu unserer Erweiterung, zu »Sprite+Grafik-Basic«. Allein für Sprites gibt's 32 neue Befehle. Außer den üblichen Funktionen, die ein Sprite entstehen lassen, sind die tollsten Parameter, wie Spiegeln, Drehen und vieles mehr kein Programmieraufwand mehr. Durch vielseitige Befehle zur Sprite-Erzeugung kann

UNMÖGLICH

selbst der blutigste Laie ohne zusätzlichen Editor seine Computerwesen erschaffen. Die Sprite-Blöcke (max. 144) benötigen selbstverständlich keinen Basic-Speicher.

Aber auch für den richtigen Bildschirmhintergrund stehen 14 neue Befehle parat. Scrollen in vier Richtungen und Auffüllen mit beliebigen Zeichen wird zum Kinderspiel.

Natürlich gibt es jede Menge neuer Befehle für hochauflösende und Multicolor-Grafik. Mit Ihnen erzielen Sie Ergebnisse, die man sonst nur in professionellen Programmen sieht:

Punkte setzen oder löschen, Linien ziehen, Ellipsen und Ellipsenbögen, Radien, Rechtecke und Blöcke lassen sich mit einfachsten Befehlen definieren. Eine Besonderheit ist dabei hervorzuheben - der GPRINT-Befehl. Er ermöglicht annähernd so komfortabel wie »PRINT«, Texte in Grafiken zu plazieren. Natürlich lassen sich die Grafiken laden und speichern.

Für übersichtliche Programmstrukturen sind zehn neue Befehle zuständig. Sie sind allesamt aus höheren Basic-Dialekten bekannt:

REPEAT ... UNTIL, CASE ... THEN ... ELSE und Label-gebundene Sprungbefehle. Damit sich diese platzsparend und einfach in Ihr Programm einbinden lassen, gibt's 20 weitere Befehle - zum Editieren oder Suchen nach Zeichenketten, zeilenweisen Abspeichern oder sogar zum Nachladen von Programmen.

Wenn Sie nun Geschmack an dieser Erweiterung gefunden haben, sollten Sie die beiliegende Diskette ins Laufwerk einlegen und mit

```
LOAD"S+G-BASIC",8
```

das Programm laden. Danach starten Sie mit RUN. Eine Einschaltmeldung zeigt Ihnen die Aktivierung der Befehlserweiterung. Danach stehen Ihnen die folgenden Befehle zusätzlich zur Verfügung:

Befehle für Blockgrafik

... erlauben bei normaler Bildschirmauflösung von 25 Zeilen x 40 Zeichen die komfortable Ausgabe von Zeichen und Zeichenketten. Dabei sind x- (x-Achse) Werte von 0 bis 39, für y (y-Achse) 0 bis 24 erlaubt. Für Farben (f) sind die im Handbuch beschriebenen Farbnummern 0 bis 15 zulässig. Ein Zeichen (c) entspricht den ASCII-Werten 0 bis 255.

AT x,y,string
Gibt einen String an der Bildschirmposition x, y aus.

SET c,f,x,y
Setzt ein Zeichen (ASCII-Code c) mit der Farbe (f) an die Position x,y.

FSET f,x,y
Ändert die Farbe (f) eines Zeichens an der Bildschirmposition x,y.

ZSET c,x,y
Gibt das Zeichen c an der Position x,y aus.

COLOUR rf,hf
Setzt Rahmenfarbe (rf) und Hintergrundfarbe.

CURSOR f
Ändert die Farben der nachfolgenden Print-Befehle auf den Wert »f«

FILL x1,y1,x2,y2,c,f
Füllt eine Fläche von links oben x1,y1 bis x2,y2 mit dem Zeichen »c« und der Farbe »f«.

ZFILL x1,y1,x2,y2,c
Funktioniert wie »FILL«, nur wird die Zeichenfarbe in ihrem Originalwert belassen.

INV x1,y1,x2,y2
Invertiert alle Zeichen im angegebenen Bereich. Das heißt die Zeichen erscheinen dunkel auf hell, wenn sie vorher hell auf dunkel waren und umgekehrt.

FFILL x1,y1,x2,y2
Färbt den angegebenen Zeichenbereich mit der aktuellen Zeichenfarbe

UP x1,y1,x2,y2
Scrollt den angegebenen Bereich nach oben. Mit »DOWN« wird nach unten, mit »LEFT« nach links und mit »RIGHT« nach rechts gescrollt.

FLASH-Befehle

... machen farbliche Bereiche durch Blinken (Farbvertauschen) kenntlich. Um diese Funktion in Betrieb zu nehmen, muß zuerst auf eine neue IRQ-Routine umgeschaltet werden. Dabei sind für die Farben (f1 bis f3) die im Handbuch beschriebenen Farbnummern 0 bis 15 erlaubt. Die Blinkgeschwindigkeit (speed) wird in 1/10 sek. zwischen 1 und 255 geschaltet.

VEC
Schaltet auf die neue Interrupt-Routine um.

OLDVEC
Wechselt zurück zum alten IRQ. Die neue Routine benötigt eine gewisse Ausführungszeit. Daher sollte nach Beendigung der Flash-Befehle grundsätzlich mit »OLDVEC« zurückgeschaltet werden.

BFLASH f1,f2,speed
Der Rahmen wechselt ständig zwischen den Farben f1 und f2 mit der Blinkgeschwindigkeit speed.

GFLASH f1,f2,speed
Funktioniert wie »BFLASH«, nur wechselt der Bildhintergrund.

BGFLASH f1,f2,speed
Entspricht »BFLASH«, jedoch wechseln Hintergrund und Bildhintergrund.

FOFF
Schaltet B/G/BGFLASH wieder aus.

FLASH f1,f2,f3
Invertiert die Zeichen mit den Farben f1,f2 und f3 in ständigem Wechsel.

OFF
»FLASH« wird ausgeschaltet. Eine der Farben f1, f2 oder f3 bleibt erhalten.

Sprite-Befehle

... erlauben die komfortable Behandlung von Sprites. Als »adresse« wird dabei die Anfangsadresse des Sprite-Zeigers bezeichnet (32768+(64 x Spriteblock)). Zu Verfügung stehen Block 16 bis 31, und falls Sie keine hochauflösende Grafik bearbeiten, zusätzlich Block 128 bis 255. Die Sprite-Nummer (sn) bezeichnet das eingeschaltete Sprite. Da der C64 max. acht Sprites zuläßt, sind die Nummern 0 bis 7 erlaubt.

SCLR adresse
Löscht einen Spriteblock ab »adresse«.

SRVS adresse
Invertiert den Spriteblock (adresse).

BMOVE adresse1, adresse2
Kopiert Spriteblock 1 in Spriteblock 2.

SON sn
Schaltet das Sprite mit der Nr. »sn« ein.

SOFF sn
Schaltet das Sprite »sn« aus.

SBLOCK sn,block
Weist dem Sprite (sn) die Blocknummer (block) zu.

SMODE sn,modus
Das Sprite »sn« ist mit »modus = 0« in Multicolor, mit »1« normal dargestellt

SCOLOUR sn,f1
Das Sprite »sn« erhält die Farbe »f1« (0 bis 15).

MCOLOUR f2,f3
Teilt allen Multicolor-Sprites gemeinsam die Farben »f2« und »f3« zu.

SPRIOR sn,modus
Erteilt die Hintergrundpriorität (modus) für das Sprite »sn«. Zwei Werte sind dabei erlaubt:

»0« - das Sprite liegt über dem Hintergrund (Blockgrafik oder Hires).

»1« - der Hintergrund erscheint über einem Sprite.

XEXP sn,modus
»1« als Modus vergrößert das Sprite »sn« in der x-Achse (Breite), »0« schaltet auf normale Größe.

YEXP sn,modus
Funktioniert wie »XEXP«, gilt aber für die Höhe.

SSET sn,x,y

Setzt das Sprite »sn« an die Position x (horizontal, 0 bis 511) und y (vertikal, 0 bis 255).

SUP sn,sw

Bewegt das Sprite »sn« um die Schrittweite »sw« (0 bis 255) nach oben (up).

SDOWN sn,sw

Funktioniert wie »SUP«, bewegt aber nach unten.

SLEFT sn,sw

Entspricht »SUP«, die Richtung ist links.

SRIGHT sn,sw

Hat die gleichen Parameter wie »SUP«, bewegt aber nach rechts.

SPLOT adresse,x,y,modus, zm

In den Speicherblock mit »adresse« (32768+(64 x Sprite-Block)) wird an der x-Koordinate (multicolor 0 bis 11, normal 0 bis 23) und y-Koordinate (0 bis 20) ein Punkt gelöscht, gesetzt oder invertiert. Dabei muß in »modus« angegeben werden, ob es in multicolor (1) oder normal (0) konstruiert ist. »zm« bezeichnet den Zeichenmodus:

tzm	normal	multicolor
t0	löschen	löschen
t1	setzen	Farbe f1 setzen
t2	invertieren	Farbe f2 setzen
t3	-	Farbe f3 setzen

Befehle für komfortable Sprite-Behandlung

komfortable Sprite-Behandlung

DESIGN adresse,y,modus, string

Ermöglicht die zeilenweise Konstruktion eines Sprites ab »adresse«. Mit »y« (0 bis 20) wird die Zeile gekennzeichnet, und »modus« bezeichnet die Sprite-Art (0 = normal, 1 = multicolor). In »string« wird die Zeichenmatrix definiert. Wenn Sie »string« direkt hinter den Befehl schreiben, muß er in Gänsefüßchen stehen, ansonsten läßt sich eine String-Variable verwenden (z.B. A$). Im Normalmodus sind 24 Zeichen gefordert, bei multicolor zwölf. Dabei sind folgende Zeichen vereinbart:

Zeichen	normal	multicolor
A	setzen	Farbe f1 setzen
B	keine Änderung	Farbe f2 setzen
C	-	Farbe f3 setzen
D	-	keine Änderung
.	löschen	löschen
I	invertieren	invertieren

SCORD sn,v1,v2

Übergibt die Bildschirmposition von Sprite »sn« (0 bis 7) an v1 (x-Koordinate) und v2 (y-Koordinate). Für v1 und v2 müssen nichtindizierte, reelle Variablen eingesetzt werden, beispielsweise A, B, C1 usw. Nicht erlaubt sind z.B. A%, A(0), A$ usw.

SMOVE sn,x1,y1,x2,y2, sw,s

Bewegt ein Sprite (sn) vom Ausgangspunkt (x1,y1) zu den Koordinaten x2 und y2 in der Schrittweite »sw« (1 bis 255). Die Geschwindigkeit (s) darf dabei Werte von 1 bis 255 annehmen (1 = schnell, 255 = langsam)

MOVETO sn,x,y,sw,s

Funktioniert wie »SMOVE«, nur wird von der derzeitigen Position bis zu x und y bewegt.

RRIGHT adresse

Scrollt die Sprite-Matrix ab »adresse« nach rechts.

RLEFT adresse

Entspricht »RRIGHT«, hat aber die Scroll-Richtung links.

RUP adresse

Funktioniert wie »RRIGHT«, die Scroll-Richtung ist aufwärts.

RDOWN adresse

Die »adresse« ist wie bei »RRIGHT«, die Scroll-Richtung aber abwärts.

XMIR modus,adresse

Spiegelt die Sprite-Matrix ab »adresse« in x-Richtung. Mit »modus« muß mitgeteilt werden, ob das Sprite im Multicolor- (1) oder im Normalmodus (0) gespiegelt wird.

XMIR adresse

Eine Sprite-Matrix ab »adresse« wird in y-Richtung gespiegelt.

TURN modus,adresse

Dreht die Sprite-Matrix ab »adresse« um 180 Grad. Auch hier muß im »modus« mitgeteilt werden, ob das Sprite multicolor (1) oder einfarbig (0) ist.

ROTR adresse

Rotiert einen Sprite-Block um 90 Grad nach rechts. Achtung: Funktioniert nur für einfarbige Sprites, arbeitet nicht in multicolor.

ROTL adresse

Dreht einen Sprite-Block um 90 Grad nach links. Achtung: Funktioniert nur für einfarbige Sprites, arbeitet nicht zusammen mit Multicolor.

DETECT modus

Setzt die Kollisionsregister zurück. Bei modus = 0 wird das Sprite-Sprite-, bei modus = 1 das Sprite-Hintergrundregister zurückgesetzt. Die Abfrage der Kollisionen finden Sie unter »Funktionen (CHECK)«.

Befehle für die hochauflösende Grafik

... behandelt sowohl Hires- als auch Multicolor-Grafik. Dabei sind folgende Werte erlaubt:

Hires-Grafik	Multicolor-Grafik
x-Achse, 0 bis 319	x-Achse, 0 bis 159
y-Achse, 0 bis 199	y-Achse, 0 bis 199

Werte, die über diesen Bereich hinausgehen, werden ignoriert. Für den Zeichenmodus (zm) ist folgendes vereinbart:

»zm«	Hires-Grafik	Multicolor-Grafik
0	löschen	löschen
1	setzen	Farbe f1 setzen
2	invertieren	Farbe f2 setzen
3	-	Farbe f3 setzen
4	-	invertieren

Wenn Sie Sprites zusammen mit Multicolor und Hires verwenden, dürfen die Blocks 128 bis 255 (40960 bis 49088) nicht verwendet werden. Die Sprites liegen genau im Grafikspeicher und würden am Bildschirm ein wirres Pixelmuster erzeugen.

HRG pf,hf

Schaltet die hochauflösende Grafik ein und setzt Punktfarbe (pf) und Hintergrundfarbe (hf).

NRM

Wechselt auf normale Blockgrafik/Text zurück. Dabei wird der Bildschirm gelöscht und die Hintergrund- und Rahmenfarbe wie bei <RUN/STOP RESTORE> gesetzt.

CLS

Löscht den Grafikbildschirm und setzt die Farben wie unter »HRG« festgelegt.

MULTI f1,f2,f3

Der Multicolor-Modus wird eingeschaltet mit den Farben f1, f2 und f3 (0 bis 15, siehe Handbuch).

MODE zm

Stellt den Zeichenmodus (zm) ein (siehe oben). Dieser ist für alle nachfolgenden Grafikbefehle gültig.

PLOT x,y

Setzt einen Punkt an die Koordinaten x (horizontal) und y (vertikal).

DRAW x,y

Zieht eine Linie vom zuletzt gesetzten Punkt zu den Koordinaten x und y.

ASCII-Code	Tastenkombination	Funktion
1	CTRL + A	Schriftgröße in x-Richtung +1
2	CTRL + B	Schriftgröße in x-Richtung —1
3	CTRL + C	Schriftgröße in y-Richtung +1
4	CTRL + D	Schriftgröße in y-Richtung —1
22	CTRL + V	Cursor 1 Pixel nach links
16	CTRL + P	Cursor 1 Pixel nach rechts
15	CTRL + O	Cursor 1 Pixel nach oben
21	CTRL + U	Cursor 1 Pixel nach unten
157	SHIFT + CRSR lr	Cursor left
29	CRSR lr	Cursor right
145	SHIFT + CRSR ud	Cursor up
17	CRSR ud	Cursor down
19	HOME	Cursor home
147	CLR/HOME	siehe CLS
13	CTRL + M	Carriage Return
14	CTRL + N	Umschaltung Kleinbuchstaben
142	keine	Umschaltung Großbuchstaben
18	CTRL RVSON	RVS on
146	CTRL RVSOFF	RVS off
6	CTRL + F	Zeichenmodus 0
7	CTRL + G	Zeichenmodus 1
10	CTRL + J	Zeichenmodus 2
11	CTRL + K	Zeichenmodus 3
12	CTRL + L	Zeichenmodus 4

Tabelle 1. Die Steuerzeichen unter »GPRINT«

LINE x1,y1,x2,y2

Zeichnet eine Linie von der x- und y-Position (x1,y1) bis zu den Koordinaten x2, y2.

REC x1,y1,x2,y2

Erzeugt ein Rechteck mit der linken oberen Ecke x1, y1 und der rechten unteren Ecke x2, y2.

BLOCK x1,y1,x2,y2

Funktioniert wie »REC«, nur wird das Rechteck ausgefüllt.

ANGL x,y,rx,ry,w

Zeichnet eine Gerade, die dem Radius einer Ellipse mit dem Startpunkt »x, y«, der Breite »rx« und der Länge »ry« entspricht. Der Winkel (w) ist vom Mittelpunkt der Ellipse aus festgelegt.

ARC x,y,rx,ry,sw,ew,a

Erzeugt einen Ellipsenbogen mit dem Mittelpunkt »x, y«, der Breite »rx« und der Länge »ry«. Zusätzlich müssen Startwinkel (sw) und Endwinkel (ew) angegeben werden (0 bis 360 Grad). Der Winkelabstand (a) der berechneten Punkte (1 bis 360 Grad) hat Einfluß auf die Zeichengeschwindigkeit und Glättung. Mit größeren Winkeln lassen sich Polyeder konstruieren.

GPRINT string

Gibt einen in »string« enthaltenen Text am Grafikbildschirm aus. Schriftgröße, Zeichensatz, Farbe und Cursor-Steuerung lassen sich durch Steuerzeichen beeinflussen (Tabelle 1).

GSAVE gn,name

Speichert die hochauflösende Grafik auf das Gerät mit der Nummer »gn« unter dem Namen »name«.

GLOAD gn,name

Lädt hochauflösende Grafik vom Gerät mit der Nummer »gn« und mit dem Namen »name«.

Achtung:

Bei allen LOAD/SAVE-Vorgängen werden die Parameter von GPRINT zerstört. Ein Blinken der Floppy-LED ist normal. Bei manchen Floppies läßt sich der LOAD/SAVE-Befehl nicht im Programmodus anwenden. Der Name ist bei der direkten Anweisung in Gänsefüßchen zu setzen.

Editorbefehle

... dienen zur einfacheren Eingabe von Programmen. Sie sollten, obwohl auch im Programm anwendbar, nur im Direktmodus gebraucht werden. Die meisten dieser Befehle beenden ein bestehendes Programm mit »READY«.

FKEY fn,string

Weist einer Funktionstaste (fn) eine Zeichenkette (string) mit max. 20 Zeichen zu. Die Funktionstasten neun bis zwölf sind über die CBM-Taste erreichbar. Bevor Sie diese Funktion erreichen, muß der IRQ mit »VEC« aktiviert werden.

DISP

Gibt die Belegung der Funktionstasten aus.

PAGE zn

Listet zehn Basic-Zeilen nach der Zeilennummer »zn«.

MOVE

Die nächsten zehn Basic-Zeilen (nach »PAGE«) werden am Bildschirm ausgegeben.

BACK

Gibt zehn Zeilen vor den zuletzt angezeigten aus.

OLD

Ein durch »NEW« gelöschtes Programm wird wieder aktiviert.

DUMP

Gibt die Inhalte aller Variablen aus und zeigt die Funktionsnamen selbstdefinierter Funktionen an. Sie werden durch »!« gekennzeichnet.

ARRAY v

Zeigt den Inhalt aller Elemente des Feldes »v« am Bildschirm an.

MEM

Zeigt die momentane Speicherbelegung an.

MERGE name,gn

Lädt das Programm »name« vom Gerät »gn« und hängt es an ein bestehendes. Die Zeilennummern des schon vorhandenen Basics müssen niedriger sein als die des nachgeladenen.

ZSAVE gn,name,z1,z2

Speichert einen Programmteil auf das Gerät »gn« unter der Bezeichnung »name«. Dabei werden nur die Zeilennummer »z1« bis »z2« berücksichtigt.

ZLOAD z,name,gn

Lädt ein Programm, das unter »name« auf dem Gerät »gn« gespeichert sein muß, und fügt es ab der Zeile »z« im Computer an. Alle bisher auf »z« folgenden Nummern gehen dabei verloren.

DEL z1,z2

Löscht ein Programm von Zeilennummer z1 bis z2.

HELP

Gibt alle Basic-V-2.0-Befehle aus

@HELP

Erzeugt eine Liste aller G+B-Basic-Befehle.

FIND Zeichenfolge

Zeigt die Nummern der Zeilen an, in denen die Zeichenfolge enthalten ist. Werden die gesuchten Zeichen nicht in Gänsefüßchen gesetzt, sucht der Befehl nur nach den entsprechenden Variablen. Basic-Befehle werden nicht gefunden.

AUTO zn,a

Für die Basic-Eingabe erfolgt eine automatische Zeilenvorgabe mit der ersten Nummer »zn« und der Schrittweite »a«. Ist die vorgegebene Zeile schon im Programm, färbt sich der Bildschirmrahmen rot, ansonsten blau.

TRACE

In der obersten Bildschirmzeile wird die aktuelle Programmnummer angezeigt und der aktuelle Befehl revers gekennzeichnet. Dieser Befehl wird erst nach Tastendruck abgearbeitet.

RETRACE

Schaltet den »TRACE«-Modus aus.

ERROR

Gibt den zuletzt aufgetretenen Fehler und die dazugehörige Zeile aus.

COLD

Startet Sprite+Grafik-Basic neu.

Strukturierende Befehle

... machen Programme übersichtlicher und einfacher. Bei konsequenter Anwendung lassen sich Programme wesentlich verkürzen.

REPEAT: ... :UNTIL Bedingung

Führt die auf REPEAT folgenden Anweisungen solange durch, bis die auf UNTIL folgende Bedingung erfüllt ist.

CASE Bedingung THEN ... :ELSE ...

Ist die Bedingung erfüllt, werden die Befehle nach THEN ausgeführt. Im anderen Falle folgen die Befehle nach ELSE.

AGAIN ... :ELSE ...

War die Bedingung nach der letzten Anweisung richtig, wird zwischen AGAIN und ELSE abgearbeitet, sonst nach ELSE.

JUMP label

Springt an die durch »label« gekennzeichnete Programmstelle. »label« darf eine beliebige Zeichenkette sein.

LABEL label

Kennzeichnet einen Programmaufruf durch »JUMP« oder »CALL«. Trifft das Programm ohne JUMP- oder CALL-Aufruf auf ein LABEL, wird der nächste Befehl abgearbeitet.

CALL label

Ruft ein Unterprogramm des namens »label« auf. Es muß mit SUBEND beendet werden.

SUBEND

Das mit CALL aufgerufene Unterprogramm wird wieder verlassen, und es wird der Befehl nach CALL abgearbeitet.

Funktionen

... übertragen Parameter in Variablen und machen lo-

dez.-adr.	hex.-adr.	Belegung
0— 1024	0000—0400	wie im Handbuch beschrieben
1025—28671	0401—6FFF	Basic-RAM
28672—32767	7000—7FFF	Editor- und Strukturbefehle
32768—33791	8000—83FF	Video-Ram (Bildschirm)
33792—34815	8400—87FF	Spriteblöcke 16 bis 31
34816—40959	8800—9FFF	Blockgrafik- und Spritegrafikbefehle
40960—49151	A000—BFFF	HRG
49152—53247	C000—CFFF	HRG-Befehle
53248—57343	D000—DFFF	keine Veränderungen
57344—59392	E000—E7FF	Speicher für Farben der HRG

Tabelle 2. Die Speicherbelegung

gische Abfragen nach Ereignissen möglich. In der Beschreibung wird »A=«, also die Übergabe an eine numerische Variable, gezeigt. Natürlich sind alle anderen Variablennamen erlaubt (z.B. B, C1, DD usw.). Ebenso sind logische Vergleiche (IF ... THEN) und Programmsprünge (ON ... GOTO/GOSUB) erlaubt.

A=XCORD(0)

Holt die x-Koordinate in die Variable A.

A=YCORD(0)

Bringt die y-Koordinate in die Variable A.

A=CODE(x)y

Überträgt den Bildschirmcode des Zeichens an der Position x,y in die Variable »A«.

A=FARB(x)y

Nach Ausführung dieser Funktion befindet sich der Farbcode des Zeichens an der Position x,y in der Variablen »A«.

CHECK(0)sn1,sn2

Überprüft das Kollisionsregister Sprite/Sprite auf Berührung zweier Sprites (sn1 und sn2). Verwendet wird diese Funktion beispielsweise:

```
A = CHECK(0)0,1
```

Das Ergebnis der Auswertung des Kollisionsregisters zwischen Sprite 0 und Sprite 1 wird der Variablen »A« übergeben.

Der Wert der Variablen ist:

»0« die Sprites berühren sich nicht,

»1« die Sprites kollidieren.

CHECK(1)sn

Das Kollisionsregister Sprite/Hintergrund wird auf die Berührung Sprite/Hintergrundzeichen überprüft.

Als Ergebnis erhalten Sie »0« für keine Berührung oder »1« für Kollision.

A = TEST(0)x,y

Prüft, ob ein Punkt in der hochauflösenden Grafik gesetzt ist. Das Ergebnis in der Variablen (A) entspricht dem Zeichenmodus.

A = STICK(n)

Holt die Stellung des Joysticks in die Variable »A«. Als Werte sind möglich:

Ereignis	Wert
oben	1
unten	2
links	4
rechts	8
oben links	5
oben rechts	9
unten links	6
unten rechts	10
Feuer	Richtung+16

Fehlermeldungen

Da der Wortschatz von »Sprite+Grafik-Basic« bedeutend umfangreicher ist als der des Basic V 2.0, wurde es unumgänglich, auch zusätzliche Fehlermeldungen mit einzubauen:

UNTIL WITHOUT REPEAT

Tritt dann auf, wenn der Computer ohne vorausgegangenes REPEAT auf den Befehl UNTIL trifft.

LABEL NOT FOUND

Der Interpreter findet den LABEL-Namen nicht, auf den verzweigt werden soll.

SUBEND WITHOUT CALL

Ein Rücksprung mit SUBEND ist nur möglich, wenn dieses Unterprogramm mit CALL aufgerufen wird.

ILLEGAL SPRITE NUMBER

Es sind nur die Sprite-Nummern 0 bis 7 erlaubt. Ist der Wert höher, erscheint diese Fehlermeldung.

ILLEGAL SPRITE STRING

Hier enthält der String des DESIGN-Befehls einen Fehler. Beachten Sie, daß nur bestimmte Zeichen erlaubt sind.

SPRITE DESIGNING COORDINATE

Tritt auf, wenn beim Befehl SPLOT falsche x/y-Koordinaten angegeben werden.

Besonderheiten

Beim FILL/ZFILL/FFILL-Befehl (siehe Blockgrafik) ist die linke obere Ecke bindend vorgeschrieben. Wird an einem anderen Punkt begonnen, stürzt der Computer unweigerlich ab.

TRACE verträgt sich nicht mit den anderen Interrupt-Befehlen (VEC,OLDVEC, B/G/BG und FLASH). Der Grund dieser Unverträglichkeit liegt in der Umlenkung bei TRACE in einen Raster-IRQ. Er sorgt dafür, daß immer die obersten drei Zeilen in Blockgrafik angezeigt werden. Drücken Sie zur Abhilfe nach Beendigung der Fehlersuche mit TRACE die Tasten <RUN/STOP RESTORE>.

Der eingebaute Raster-Interrupt läßt sich auch ohne TRACE von Ihnen nützen. Er wird mit SYS31851 aktiviert. Zuvor müssen Sie in die Speicherstellen 31889 und 31903 jeweils die Rasterzeile POKEn (0 bis 255), ab der auf die hochauflösende Grafik geschaltet werden soll. Außerdem muß in der Speicherstelle 31929 die Rasterzeile enthalten sein (0 bis 255), ab der wieder auf den Textmodus zurückgeschaltet wird. Ein Ausschalten ist durch »NRM« oder <RUN/STOP RESTORE> möglich.

Im TRACE-Modus ist der Befehl GET nicht mehr funktionsfähig, da ständig der letzte Tastendruck gelöscht wird. Beachten Sie dies bei der Fehlersuche.

Die Funktion TEST setzt zusätzlich den Grafik-Cursor. Damit läßt sich vor GPRINT der Cursor auf die gewünschte Position bringen.

Verwenden Sie niemals @ohne Zusätze. Der Computer kann abstürzen. Das gleiche kann passieren, wenn Sie bei Befehlen einen oder mehrere Parameter vergessen.

Beim Speichern eines Hires-Bildes auf Diskette, beginnt die Floppylampe zu blinken, und die Farbinformation geht verloren.

Bei einer Fehlermeldung wird zusätzlich die zuletzt abgearbeitete Zeile gelistet. Der Befehl, bei dem der Interpreter aufgehört hat, zu arbeiten, wird invertiert dargestellt. Dadurch läßt sich in den meisten Fällen der Fehler schneller rekonstruieren.

Beachten Sie auch die gegenüber dem normalen Betriebssystem geänderte Speicherbelegung (Tabelle 2). Sie macht trotz des erweiterten Befehlssatzes den großen Basic-Speicher möglich.

Demoprogramme auf Diskette

Um Ihnen den Einstieg in Sprite+Grafik-Basic zu erleichtern, befinden sich Demoprogramme mit auf der Diskette. Achtung: Vor dem Laden muß die Basic-Erweiterung initialisiert sein! Beachten Sie dazu unsere Ladeanweisung am Anfang. Laden Sie nun die erste Grafikdemo »LINE1« und starten Sie mit RUN. Das Beispielprogramm zeigt eine Variation des LINE-Befehls mit Farbänderung und Möglichkeiten der GPRINT-Anweisung.

Das nächste Beispiel ist »LINE2«. Es demonstriert, wie man die SIN-Funktion mit LINE einsetzen kann (Abb.1).

Quadrate und Rechtecke stellen S+G-Basic vor keine Probleme: Das zeigt Ihnen unser Demo »REC«.

»ARC« unterstreicht die Möglichkeit, Kreise und Ellipsen auf den Grafikbildschirm zu zaubern (wie unser Demoprogramm beweist).

Das Beispiel »FLUGZEUG« bringt eine Verbindung von LINE und ARC (Abb.2). Die x- und y-Positionen der einzelnen Grafikpunkte wurden als Datazeilen definiert. Mit »SPRITEDEMO1« wird's lebendig auf dem Bildschirm. »SPRITEDEMO2« läßt ein Pferd über den Bildschirm laufen. Mit dem Wissen unserer Demoprogramme dürfte es ein leichtes sein, in kürzester Zeit ein Sprite- und Grafikprofi zu werden. (gr)

Kurzinfo: Sprite+Grafik-Basic

Programmart: Basic-Erweiterung
Laden: LOAD"S+G-BASIC",8
Starten: nach dem Laden RUN eingeben
Besonderheiten: über 100 neue Basic-Befehle für Sprites, Grafik und strukturierte Programmierung
Benötigte Blocks: 58
Programmautor: Jens Schwarz

XBasic Plus – über 40 neue Befehle

Der elegante Weg

Der Interpreter des C64 kennt weder Sound noch hochauflösende Grafik. Da helfen nur PEEKs und POKEs – oder unsere Erweiterung. Sie macht Schluß mit umständlicher Programmierung.

von Stephan Blietz

Das Basic des C64 ist nur mit den unverzichtbaren Befehlen ausgestattet. Dadurch wird zwar jedes Programm möglich, aber meist nur mit riesigem Speicherbedarf oder nicht tolerierbarem Zeitaufwand in der Ausführung. Hier ist »XBasic Plus« nützlich. Über 40 neue Befehle stehen für Sie bereit.

Sie laden die Erweiterung mit der Anweisung:

```
LOAD"XBASIC+",8
```

und starten RUN. Danach meldet sich die Erweiterung mit einem Einschaltbild.

Bei der folgenden Beschreibung der Befehle deuten eckige Klammern an, daß es sich um einen optionalen Parameter handelt. Die eckigen Klammern sind nicht einzugeben. Im Abschnitt Fehler wird erläutert, welche Fehlermeldungen auftreten können. Ein »SYNTAX ERROR« bleibt unerwähnt, wenn er nur durch eine falsche Schreibweise begründet ist. Das gleiche gilt für den »ILLEGAL QUANTITY ERROR«, der eine Bereichsüberschreitung der Befehlsparameter anzeigt.

Programmierhilfen

AUTO [Schrittweite]

Nach Eingabe einer Programmzeile wird automatisch eine um Schrittweite erhöhte Zeilennummer ausgegeben. Der AUTO-Modus wird durch Eingabe einer Leerzeile wieder verlassen. »Schrittweite« darf Werte zwischen 0 und 255 annehmen. »0« ist zwar möglich, aber nicht sinnvoll, da immer die gleiche Zeilennummer erscheint. Wird »Schrittweite« weggelassen, so findet eine Erhöhung um 10 statt.

Fehler: SYNTAX ERROR – es wurde eine ungültige Zeilennummer erzeugt (64000).

```
AUTO 200
100 IF A THEN PRINT "HALLO"
```

Nach Eingabe obiger Zeilen wird die Zeilennummer 300 automatisch vorgegeben.

DELETE [Anfang] [Ende]

....löscht den durch »Anfang« und »Ende« spezifizierten Programmzeilenblock. »Anfang« gibt die Nummer der ersten Zeile und »Ende« die der letzten Zeile des Blocks an. Ist »Anfang« die erste Zeile des Programms bzw. »Ende« die letzte Zeile des Programms, kann der jeweilige Parameter weggelassen werden. Im übrigen gilt der übliche Wertebereich für Basic-Programmzeilen. Werden Zeilennummern von nicht existenten Programmzeilen angegeben oder wird DELETE ohne Parameter eingegeben, so hat der Befehl keine Wirkung.

Fehler: SYNTAX ERROR – es wurde eine ungültige Zeilennummer (64000) als Parameter übergeben oder »Anfang« ist größer als »Ende«.

```
100 FOR I=1 TO 100
110 PRINT I
120 NEXT
DELETE 110-
```

Nach Ausführung von DELETE sind die Zeilen 110 und 120 gelöscht.

DUMP

....gibt die Namen und Werte aller nicht indizierten Variablen, sowie die Namen mit DEFFN definierter Funktionen aus. Die Reihenfolge der Ausgabe entspricht der Speicherfolge. Zur Kennzeichnung werden Integervariable durch »%« und Stringvariable durch »$« markiert. Funktionsbezeichner werden durch »*« kenntlich gemacht. Achtung: Bei einer großen Menge anzuzeigender Variablen kann es sein, daß der Bildschirm durchscrollt. Um zu verhindern, daß Information verlorengeht, leiten Sie am besten die Ausgabe vorher auf einen Drucker um.

```
10 A = 10.457
20 C% = 16
30 T$ = "HALLO"
40 DEF FN G(X) = 2*X+3
```

Der Befehl DUMP hat nach Ausführung dieses Programmfragments folgende Ausgabe zur Folge:

```
A = 10.457
C% = 16
T$ ="HALLO"
G*
X = 0
```

MEMORY

...zeigt den momentanen Speicherbedarf von Programm und Variablen und den freien Speicher.

```
10 A = 16.6
20 C$ = "TEST"
30 FOR T=1 TO 100: PRINT T: NEXT
```

Nach Abarbeiten des obigen Programmstücks erzeugt MEMORY folgende Ausgabe:

```
Programm: 55
Variable: 21
Arrays   : 0
Strings  : 0
Free     : 30643
```

Beachten Sie, daß trotz einer String-Zuweisung im Programm kein String-Speicherplatz belegt wurde, da der Wert des Strings vollständig im Programmtext abgelegt ist!

RENUMBER [[*Anfang*],*Schrittweite*]

....numeriert das im Speicher befindliche Basic-Programm neu. »Anfang« gibt dabei die erste Zeilennummer an, während »Schrittweite« den Abstand der Zeilennummern (siehe AUTO) festlegt. Die Parameter dürfen alle Werte von gültigen Zeilennummern annehmen. Mit RENUMBER ohne Parameter wird das Programm mit erster Zeilennummer 10 und Schrittweite 10 umnumeriert. Ohne »Anfang« werden die neuen Zeilennummern von 0 beginnend erzeugt. RENUMBER verändert auch die Parameter der Befehle GOTO und GOSUB entsprechend, jedoch nur wenn keine berechneten Zeilennummern als Sprungziele angegeben sind (siehe GOSUB/GOTO). Um eine (falsche!) Anpassung dieser Parameter zu verhindern, ist der dem Befehl folgende Ausdruck in Klammern zu setzen (»GOTO(200+6*5)«) und die Anpas-

64'er Sonderhefte alle auf einen Blick

Die 64'er Sonderhefte bieten Ihnen umfassende Information in komprimierter Form zu speziellen Themen rund um die Commodore C 64 und C 128. Ausgaben, die eine Diskette enthalten, sind mit einem Diskettensymbol gekennzeichnet.

C 64, C 128, EINSTEIGER



SH 0022: C 128 III
Farbiges Scrolling im 80-Zeichen Modus / 8-Sekunden-Kopierprogramm



SH 0026: Rund um den C64
Der C64 verständlich für Alle mit ausführlichen Kursen



SH 0029: C 128
Starke Software für C 128/ C 128D / Alles über den neuen C 128D im Blechgehäuse




SH 0036: C 128
Power 128: Directory komfortabel organisieren / Haushaltsbuch: Finanzen im Griff / 3D-Landschaften auf dem Computer



SH 0038: Einsteiger
Alles für den leichten Einstieg / Super Malprogramm / Tolles Spiel zum selbermachen / Mehr Spaß am Lernen




SH 0050: Starthilfe
Alles für den leichten Einstieg / Heiße Rythmen mit dem C 64 / Fantastisches Malprogramm



SH 0051: C 128
Volle Floppy-Power mit "Rubikon" / Aktienverwaltung mit "Börse 128"



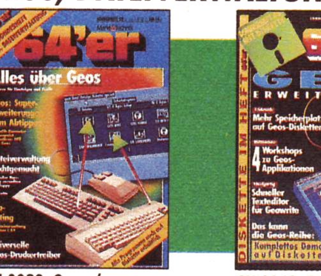
SH 0058: 128er
Übersichtliche Buchhaltung zuhause / Professionelle Diagramme

SH 0062: Erste Schritte
RAM-Exos: Disketten superschnell geladen/ Exbasic Level II: über 70 neue Befehle/ Rafinessen mit der Tastatur



SH 0064: 128ER
Anwendungen: USA Journal / Grundlagen: CP/M, das dritte Betriebssystem/ VDC-Grafik: Vorhang auf für hohe Auflösung

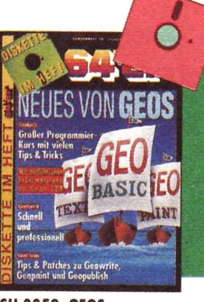
GEOS, DATEIVERWALTUNG



SH 0028: Geos / Dateiverwaltung
Viele Kurse zu Geos / Tolle Geos-Programme zum Abtippen



SH 0048: GEOS
Mehr Speicherplatz auf Geos-Disketten / Schneller Texteditor für Geowrite / Komplettes Demo auf Diskette



SH 0059: GEOS
GeoBasic: Großer Programmierkurs mit vielen Tips & Tricks

PROGRAMMIERSPRACHEN




SH 0035: Assembler
Abgeschlossene Kurse für Anfänger und Fortgeschrittene



SH 0040: Basic
Basic Schritt für Schritt / Keine Chance für Fehler / Profi-Tools und viele Tips

ANWENDUNGEN



SH 0031: DFÜ, Musik, Messen-Steuern-Regeln
Alles über DFÜ / BTX von A-Z / Grundlagen / Bauanleitungen



SH 0046: Anwendungen
Das erste Expertensystem für den C 64 / Bessere Noten in Chemie / Komfortable Dateiverwaltung



SH 0056: Anwendungen
Gewinnauswertung beim Systemlotto / Energieverbrauch voll im Griff / Höhere Mathematik und C64

TIPS, TRICKS & TOOLS



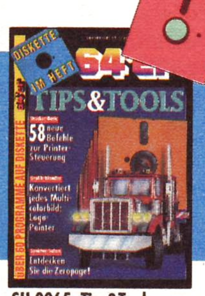

SH 0024: Tips, Tricks& Tools
Die besten Peeks und Pokes sowie Utilities mit Pfiff



SH 0043: Tips, Tricks&Tools
Rasterinterrupts - nicht nur für Profis / Checksummer V3 und MSE / Programmierhilfen



SH 0057: Tips & Tricks
Trickreiche Tools für den C64 / Drucker perfekt installiert



SH 0065: Tips&Tools
Streifzug dirch die Zeropage/ Drucker-Basic: 58 neue Befehle zur Printer-Steuerung/ Multicolorgrafiken konvertieren/ über 60 heiße Tips&Tricks

HARDWARE



SH 0025: Floppylaufwerke
Wertvolle Tips und Informationen für Einsteiger und Fortgeschrittene



SH 0032: Floppylaufwerke und Drucker
Tips&Tools / RAM-Erweiterung des C64 / Druckerroutinen



SH 0047: Drucker, Tools
Hardcopies ohne Geheinmisse / Farbige Grafiken auf s/w-Druckern




SH 0067: Wetterstation:
Temperatur, Luftdruck und feuchte messen/ DCF-Funkuhr und Echtzeituhr/ Daten konvertieren: vom C64 zum Amiga, Atari ST und PC

DTP



SH 0039: DTP, Textverarbeitung
Komplettes DTP-Paket zum Abtippen / Super Textsystem / Hochauflösendes Zeichenprogramm

64'er Magazin auf einen Blick

Diese 64'er-Ausgaben bekommen Sie noch bei Markt&Technik für jeweils 7,- DM. Die Preise für Sonderhefte und Sammelbox entnehmen Sie bitte dem Bestellcoupon. Tragen Sie Ihre Bestellung im Coupon ein und schicken Sie ihn am besten gleich los, oder rufen Sie einfach unter 089 - 20 25 15 28 an.

6/90: Programmierung: endlich Basic 3.5 für C64 / Softwaretest: die besten Fußballprogramme / Videostudio, C64 in Börsenfieber

7/90: Extratouren: CD-Musicbox mit C64 und Bauanleitung Pulsmesser / Sammelposter C64 im Riesenformat

9/90: Großer C64-Reparaturkurs / Faszination: Amateurfunk / Neuigkeiten aus der GEOS-Welt / Super-Spiele zum Abtippen

10/90: Bauanleitungen: 5 Wochenend-Projekte / ECOM-das Super-Basic / Test: Die besten Drucker unter 1000 DM / C64-Reparaturkurs

11/90: Bausatztest: Der Taschengeldplotter / Vergleichstest: Drucker der Spitzenklasse / 5 Schnellbauschaltungen

12/90: Abenteuer BTX / Multitasking für C64 / Großer Spieleschwerpunkt / Programmierwettbewerb: 30 000 DM zu gewinnen

1/91: Die Besten Tips&Tricks / Neu: Reparaturecke / Floppy-Flop: Betriebssystem überlistet / Jahresinhaltsverzeichnis

2/91: Sensation: Festplatte für den C 64 / Drucken ohne Ärger / Listing des Monats: Actionspiel "Ignition" / Longplay: Dragon Wars

3/91: Bauanleitung: universelles Track-Display / Alles über Module für den C 64 / Festplatte HD 20 unter GEOS

4/91: Spiele-Schwerpunkt: 100 Tips, News, Tests / Neu: Grafikkurs / Fischer-Baukästen / Bauanleitung: Digitizer

5/91: Ätzanlage unter 50.- DM / GRB-Monitor am C64 / Longplay: Bard's Tale / Reparaturkurs: Die neuen C64 / Piratenknacker

6/91: C64er-Meßlabor: universell Erweiterungsfähig / Test: Pocket-Wrighter 3.0 - Bestes C64 Textprogramm / Listing des Monats: Autokosten im Griff

7/91: Trickfilm mit dem C 64 / Bauanleitung: 1541-Floppy mit Batteriebetrieb / Listing des Monats: Basic-Butler

8/91: Drucker unter 1000 DM / Test: GEO-RAM / Listing des Monats: 80-Farben-Malprogramm / Longplay: Secret of the Silver Plate

GRAFIK



SH 0020: Grafik
Grafik-Programmierung / Bewegungen



SH 0045: Grafik
Listings mit Pfiff / Alles über Grafik-Programmierung / Erweiterungen für Amica-Paint



SH 0055: Grafik
Amica-Paint: Malen wie ein Profi / DTP-Seiten vom C64 / Tricks&Utilities zur Hires-Grafik



SH 0063: Grafik
Text und Grafik mischen ohne Flimmern / EGA: Zeichenprogramm der Superlative / 3 professionelle Editoren

SH 0068: Anwendungen
Kreuzworträtsel selbstgemacht/ Happy Synth: Super-Synthesizer für Sound-Freaks/ Der C64 wird zum Planetarium/ Sir-Compact: Bit-Packer verdichtet Basic- und Assemblerprogramme

SPIELE



SH 0030: Spiele für C 64 und C 128
Spiele zum Abtippen für C 64/ C 128 / Spieleprogrammierung



SH 0037: Spiele
Adventure, Action, Geschicklichkeit / Profihilfen für Spiele / Überblick, Tips zum Spielekauf



SH 0042: Spiele
Profispiele selbst gemacht / Adventure, Action, Strategie



SH 0049: Spiele
Action, Adventure, Strategie / Sprites selbst erstellen / Viren-killer gegen verseuchte Disketten



SH 0052: Abenteuerspiele
Selbstprogrammieren: Von der Idee zum fertigen Spiel / So knacken Sie Adventures



SH 0054: Spiele
15 tolle Spiele auf Diskette/ der Sieger unseres Programmierwettbewerbs: Crillion II/ ein Cracker packt aus: ewige Leben bei kommerziellen Spielen



SH 0060: Adventures
8 Reisen ins Land der Fantasie - so macht Spannung Spaß



Top Spiele 1
Die 111 besten Spiele im Test/ Tips, Tricks und Kniffe zu heißen Games/ Komplettlösung zu "Last Ninja II"/ große Marktübersicht: die aktuellen Superspiele für den C64



SH 0061: Spiele
20 heiße Super Games für Joystick-Akrobaten/ Cheat-Modi und Trainer POKEs zu über 20 Profi-Spielen/ Krieg der Kerne: Grundlagen zur Spielerprogrammierung



SH 0066: Spiele
15 Top-Spiele mit Action und Strategie/ Mondlandung: verblüffend echte Simulation und Super-Grafik/ High-Score-Knacker: Tips&Tricks zu Action-Games

BESTELLCOUPON

Ich bestelle ____ 64er Sonderhefte Nr. ______________ _____ **DM**

zum Preis von je: 14,- DM (Heft ohne Diskette) _____ **DM**
16,- DM (Heft mit Diskette) _____ **DM**
9,80 DM (SH "Top Spiele 1") _____ **DM**
24,- DM (für die Sonderhefte 0051/ 0058/ 0064) _____ **DM**

Ich bestelle ____ 64er Magazin Nr. ____ / ____ / ____ _____ **DM**
zum Preis von je 7,- DM

Ich bestelle ____ Sammelbox(en) _____ **DM**
zum Preis von je 14,- DM

Gesamtbetrag _____ **DM**

Ich bezahle den Gesamtbetrag zzgl. Versandkosten nach Erhalt der Rechnung.

Name, Vorname

Straße, Hausnummer

PLZ, Wohnort

AC/14 19

Telefon (Vorwahl) ______________ Ich erlaube Ihnen hiermit mir interessante Zeitschriftenangebote auch telefonisch zu unterbreiten (ggf. streichen).

Schicken Sie bitte den ausgefüllten Bestellcoupon an: **64er Leserservice, CSJ, Postfach 140 220, 8000 München 5, Telefon 089/ 20 25 15 28**

sung später von Hand vorzunehmen. RESTORE, BRANCH und EXIT werden nicht angepaßt!

Fehler: ILLEGAL QUANTITY – die Numerierung würde eine ungültige Zeilennummer erzeugen. In diesem Fall wurde keine Numerierung durchgeführt.

```
10 INPUT A$
20 IF A$="T" THEN GOTO 40
30 PRINT "FERTIG"
40 END
```

Der Befehl RENUMBER 100,50 verändert die Numerierung wie folgt:

```
100 INPUT A$
150 IF A$="T" THEN GOTO 250
200 PRINT "FERTIG"
250 END
```

RESCUE

Ein Basic-Programm, das mittels NEW oder durch einen Reset gelöscht wurde, wird wiederhergestellt. Voraussetzung ist, daß nach dem Löschen des Programms keine Programmzeilen eingegeben oder Wertzuweisungen an Variable vorgenommen wurden.

Kontrollstrukturen

BRANCH
Zeilennummer

....bestimmt aus einem mittels CALL angesprungenen Unterprogramm nach »Zeilennummer«.

Das Unterprogramm gilt damit als ordnungsgemäß verlassen, es besteht nicht die Gefahr eines Stack-Überlaufs.

Fehler: MISSING LINENUMBER — BRANCH wurde ohne Zeilennummer eingegeben.

UNDEF'D STATEMENT — die angegebene Zeilennummer existiert nicht.

BRANCH WITHOUT CALL — Es erfolgte kein Unterprogrammaufruf durch CALL.

```
10 X = 15
20 CALL FORMEL
30 PRINT Y
40 END
50 PROCEDURE FORMEL
60 Y = SIN(X)
70 IF Y=0 THEN BRANCH 40
80 SUBEND
```

Das Unterprogramm FORMEL wird nach 40 verlassen, wenn Y=0.

CALL Label

...springt zu dem durch die PROCEDURE-Zeile mit dem Namen »Label« gekennzeichneten Unterprogramm. Beachten Sie, daß ein solches Unterprogramm durch SUBEND abgeschlossen sein muß, um einen Stack-Überlauf zu verhindern. Grundsätzlich dürfen nur 32 Unterprogrammaufrufe durch CALL geschachtelt werden.

Fehler: MISSING LABEL – CALL wurde ohne Sprungziel eingegeben.

LABEL NOT FOUND – das angegebene Sprungziel existiert nicht.

TOO MANY CALL... SUBEND – die Höchstzahl von unbeendeten CALL-Rufen (32) wurde überschritten.

Beispiel: siehe BRANCH

EXIT Zeilennummer

...verzweigt aus einem von GOSUB aufgerufenen Unterprogramm zu »Zeilennummer«. Wie bei BRANCH wird das Unterprogramm dadurch ordnungsgemäß verlassen, es kann auch hier kein Stack-Überlauf entstehen.

Fehler: MISSING LINENUMBER – die Zeilennummer hinter EXIT fehlt.

UNDEF'D STATEMENT – die angegebene Zeilennummer existiert nicht.

EXIT WITHOUT GOSUB – es erfolgte kein Unterprogrammaufruf durch GOSUB.

```
10 X = 5
20 GOSUB 50
30 PRINT Y
40 END
50 Y = COS(X)
60 IF Y=0 THEN EXIT 40
70 RETURN
```

Ist Y=0, wird das Unterprogramm ab Zeile 50 nach 40 verlassen.

JUMP Label

...springt zur PROCEDURE-Zeile mit dem Namen Label und setzt dort die Programmausführung fort. JUMP ist praktisch ein GOTO mit symbolischem Sprungziel.

Fehler: MISSING LABEL – es folgt kein Sprungziel.

Tolle Grafiken lassen sich mit »SET« blitzschnell programmieren. Das Beispieldemo »XBASIC.DEMO« auf der beiliegenden Disketten zeigt dies.

Kurzinfo: XBasic Plus

Programmart: Basic-Erweiterung
Laden: LOAD"XBASIC+",8
Starten: nach dem Laden RUN
Besonderheiten: über 40 neue Befehle
Benötigte Blocks: 20
Programmautor: Stephan Blietz

LABEL NOT FOUND – das angegebene Sprungziel konnte nicht gefunden werden.

```
10 INPUT A
20 IF A>10 THEN JUMP
GROESSER
30 A = 2*A
40 END
50 PROCEDURE GROESSER
60 A = 3*A
```

»Ist A größer als 10, wird in Zeile 50 gesprungen.

PROCEDURE Label

...weist einem Unterprogramm den Namen »Label« zu. JUMP und CALL rufen es aus dem Programm auf. Hinter »Label« stehende Befehle werden ignoriert. Eine Fehlermeldung erscheint nur, wenn versucht wird mit JUMP oder CALL ein nicht existierendes Label anzuspringen.

Beispiel: siehe »JUMP« und »CALL«

REPEAT [*Anweisungen*] UNTIL Bedingung

...erfüllt die Anweisungen zwischen »REPEAT« und »UNTIL« so lange, bis die Bedingung hinter UNTIL erfüllt ist. Die Schleife wird mindestens einmal ausgeführt. Ein Weglassen der Anweisung ist nur sinnvoll, wenn auf eine Bedingung gewartet werden soll, die sich unabhängig von der Programmausführung einstellen kann (z.B. das Drücken einer bestimmten Taste). Stehen REPEAT und UNTIL in einer Programmzeile, müssen sie durch Doppelpunkte von der Anweisung getrennt werden. Entsprechendes gilt, wenn der Bedingung nach UNTIL weitere Anweisungen folgen.

Beachten Sie, daß höchstens 32 REPEAT..UNTIL-Schleifen geschachtelt werden dürfen.

Fehler: TOO MANY REPEAT...UNTIL – es wurde versucht mehr als 32 REPEAT...UNTIL-Schleifen zu verschachteln.

UNTIL WITHOUT REPEAT – es wurde ein UNTIL ohne zugehöriges REPEAT entdeckt.

```
10 REPEAT
20 A=A+1
30 POKE203,64
40 REPEAT: UNTIL PEEK(603)<>64
50 UNTIL A=100
60 END
```

Das Programm erhöht den Inhalt der Variablen A um 1 und wartet dann auf einen Tastendruck. Dies wird so lange wiederholt, bis A den Wert 100 hat.

SUBEND

... beendet ein durch CALL aufgerufenes Unterprogramm. Die Programmausführung wird unmittelbar nach dem aufrufenden CALL fortgesetzt.

Fehler: SUBEND WITHOUT CALL – es wurde ohne CALL in ein Unterprogramm gesprungen.

```
10 FOR I=1 TO 10
20 CALL DRUCKEN
30 NEXT
40 END
50 PROCEDURE DRUCKEN
60 PRINT TAB(10) I
70 SUBEND
```

Die Schleifenvariable wird durch Ansprung der Routine DRUCKEN ausgegeben.

Befehle zur Steuerung von Peripheriegeräten

DIR Nummer

...gibt das Inhaltsverzeichnis der Diskette im Laufwerk Nummer aus. Die Ausgabe kann durch Festhalten der CTRL-Taste verlangsamt werden. Im Gegensatz zum Laden mit »LOAD"$",8,1« bleibt das im Speicher befindliche Programm erhalten. Nummer darf die Werte 8 und 9 annehmen.

ERROR

...liest den Fehlerkanal des Diskettenlaufwerks mit der Gerätenummer 8 aus und zeigt das Ergebnis an.

HCOPY

Eine Hardcopy des Textbildschirms im Groß-/Grafikmodus wird auf dem Drucker (Gerätenummer 4) ausgegeben. Es eignen sich alle zum C64 kompatiblen Drucker.

Befehle zur Tonerzeugung

ENVELOPE Stimme Nr,A,D,S,R

...lädt den Hüllkurvengenerator der Stimme »StimmNr« mit den Werten »A« (Attack), »D« (Decay), »S« (Sustain) und »R« (Release). »StimmNr« muß im Bereich 1 bis 3, die übrigen Parameter im Bereich 0 bis 15 liegen (s. Handbuch).

```
ENVELOPE 1,5,10,12,4
```

setzt die Hüllkurve für Stimme 1.

VOL Lautstärke

Die Lautstärke wird für alle Stimmen auf den angebenen Wert gesetzt. Sie darf Werte von 0 bis 15 annehmen.

WAVE StimmNr,Steuerwort

...lädt das Steuerregister (Register 4,11 oder 18) der durch »StimmNr« (s. ENVELOPE) spezifizierten Stimme wird mit Steuerwort geladen. Steuerwort ist eine 8-Bit-Binärzahl, die nur aus »0« und »1« bestehen darf (z.B 00100101). Die einzelnen Bits schalten die Wellenform (Rauschen, Rechteck, Sägezahn und Dreieck) und andere Parameter (Test, Ringmodulation, Synchronisation und Ein/Aus) an (»1«) und aus (»0«). Näheres hierzu finden sie auch im Sonderheft Nr. 11, zu bestellen bei CSJ, Tel: 089/20251527. Preis: 14 Mark.

Fehler: SYNTAX ERROR – das Steuerwort hat eine falsche Form (keine acht Stellen oder falsche Zeichen).

```
WAVE 2,01000001
```

...wählt Wellenform Rechteck für Stimme 2 und schaltet diese ein.

Blockgrafik (Textmodus)

BORDER Farbe

...setzt die Farbe des Bildschirmrahmens auf den Wert von »Farbe«. Farbe kann zwar zwischen 0 und 255 liegen, Werte über 15 sind aber nicht sinnvoll, da sich die 16 möglichen Farben (0 bis 15) ständig wiederholen. So entspricht »0 bis 15« dem Bereich »16 bis 31« und »32 bis 47« usw.

```
BORDER 1
```

...erzeugt einen weißen Bildschirmrahmen.

FILL x,y,Länge,Höhe,Code

...füllt ein Rechteck mit horizontaler Ausdehnung »Länge« und vertikaler Ausdehnung »Höhe« mit dem Zeichen »Code« (Bildschirmcode). Die Koordinaten der linken oberen Ecke sind »x,y«. Sollte das Rechteck nicht ganz in das Video-RAM passen, werden die überstehenden Teile abgeschnitten (ohne Fehlermeldung). »Länge« und »x« muß im Bereich von 0 bis 39 liegen, für »Höhe« und »y« ist 0 bis 24 erlaubt. »Code« liegt zwischen 0 und 255.

Beachten Sie, daß die Zeichen unsichtbar bleiben, wenn die entsprechenden Bytes im Farb-RAM mit der Hintergrundfarbe belegt sind.

FILL arbeitet auch dann korrekt, wenn das Video-RAM verschoben wurde, vorausgesetzt, das Highbyte der neuen Adresse wurde in 0288 hex abgelegt.

```
10 FILL 0,0,39,24,1
20 PAINT 0,0,39,24,2
```

Der gesamte Bildschirm wird mit schwarzen A's gefüllt (s.a. »PAINT«).

INK Farbe

... setzt die Zeichenfarbe (s. »BORDER«).

```
INK 0
```
...bewirkt, daß alle folgenden Zeichen in Schwarz ausgegeben werden.

PAINT x,y,Länge,Höhe,Farbe
...färbt ein Rechteck mit dem Wert von »Farbe«. Die Parameter entsprechen denen von »FILL« mit Ausnahme von »Farbe« (s. »BORDER«). FILL hat nur eine sichtbare Wirkung wenn der Bildschirm nicht gelöscht ist.
Beispiel: s. »PAINT«
PAPER Farbe
...setzt die Hintergrundfarbe (s. »BORDER«).

```
PAPER 6
```
...erzeugt einen blauen Hintergrund.

Hochauflösende Grafik

Im hochauflösenden Modus verschiebt sich das Video-RAM, das als Farb-RAM dient, nach $C000 (49152) und die Hires-Bitmap liegt ab $E000 (57344).
CLEAR
Der Grafikbildschirm wird gelöscht.
HRG Punkt,Hintergrund
...schaltet den VIC in den hochauflösenden Single-Color-Grafikmodus und initialisiert das Farb-RAM mit den Werten »Punkt« für die Punktfarbe und »Hintergrund« für die Hintergrundfarbe. Die beiden Parameter dürfen Werte von 0 bis 15 annehmen. Nach Beenden eines Programms kehrt der Rechner automatisch in den Textmodus zurück.

```
HRG 1,0
```
...schaltet in den hochauflösenden Grafikmodus und setzt Weiß als Punktfarbe und Schwarz als Hintergrundfarbe.
INVERS x,y
...invertiert den Grafikpunkt mit den Koordinaten »x« (0 bis 319) und »y« (0 bis 199).

```
10 HRG 6,14
20 CLEAR
30 FOR X=0 TO 319: INVERS X,100: NEXT
```
...zieht eine dunkelblaue Linie auf hellblauem Hintergrund.
RESET x,y
...löscht den Grafikpunkt mit den Koordinaten »x,y« (Parameter s. »INVERS«).

```
40 FOR X=0 TO 319: RESET X,
100: NEXT
```
...löscht die beim Beispiel INVERS gezogene Linie.
REVERS
Der gesamte Grafikschirm wird invertiert.
SET x,y
...setzt einen Grafikpunkt mit den Koordinaten »x,y«. (Parameter s. »INVERS«).

```
10 HRG 1,0
20 CLEAR
30 FOR X=100 TO 220: SET X,50: SET X,150: NEXT
40 FOR Y=50 TO 150: SET 100,Y: SET 220,Y: NEXT
```
Am Bildschirm wird ein weißer Rahmen auf schwarzem Grund gezeichnet (s. a. 'XBASIC.Demo' auf Diskette und Abb.).
TEXT
...schaltet den VIC zurück in den Textmodus.

Zeichensatzveränderung

Beim Umschalten auf den eigenen Zeichensatz verschiebt sich das Video-RAM nach $C400 (50176). Gleichzeitig ändert sich auch die Lage des 16 KByte großen Adreßbereich des VIC im Hauptspeicher auf $C000 (49152). Beachten Sie dies bei den Sprite-Befehlen. Die Blockgrafikbefehle stellen sich automatisch auf die neuen Bereiche ein.

CHRCOPY
...kopiert den Commodore-Zeichensatz vom ROM ins RAM ab $C800 (51200). Dort kann er jetzt durch die neuen Befehle modifiziert werden. Wenn ein Zeichensatz komplett neu kreiert wird, kann CHRCOPY auch weggelassen werden.
Beispiel: siehe CREATE
CREATE Code,Byte1,Byte2,...,Byte8
...definiert das Zeichen mit dem Bildschirmcode »Code« im neuen Zeichensatz durch Byte 1 bis Byte 8 neu. Alle Parameter dürfen den Wertebereich 0 bis 255 annehmen.

```
10 CHRCOPY
20 CREATE 1,0,0,255,126,60,24,0,0
30 RAM
```
Der Originalzeichensatz wird ins RAM kopiert und anschließend das Zeichen mit dem Bildschirmcode 1 (A) modifiziert. Die Byte-Folge definiert einen Pfeil nach unten.
RAM
...schaltet vom ROM-Zeichensatz auf den im RAM liegenden um.
Beispiel: siehe CREATE

Sprite-Grafik

Achtung beim Parameter »Block«:
Er gibt an, aus welchen Speicherplätzen die Daten zur Darstellung eines Sprites genommen werden. Ein Block ist immer 64 Byte groß. Da der VIC nur 16 KByte adressieren kann, sind 256 Blöcke (0 bis 255) möglich. Im Textmodus dürfen die Blöcke 0 bis 10 nicht benutzt werden, da hier wichtige Adressen des Betriebssystems liegen. Die Startadresse eines Blocks errechnet sich einfach aus OFFSET + 64 * Blocknummer. OFFSET hängt von der aktuellen Lage des VIC-Adreßraums in Speicher ab. Im Textmodus ist OFFSET 0. Benutzt man hochauflösende Grafik oder den RAM-Zeichensatz, so hat OFFSET den Wert 49152.
KILL [Nummer]
...schaltet das Sprite »Nummer« aus. KILL ohne Parameter wirkt für alle Sprites. »Nummer« muß im Bereich 0 bis 7 liegen.
Beispiel: siehe SPRITE
MOBEX Nummer,Richtung
...dehnt das Sprite »Nummer« in »Richtung« aus. Der Wertebereich von Nummer ist 0 bis 7; Richtung darf Werte von 0 bis 3 annehmen:
0 - keine Expansion, normale Größe
1 - Expansion in x-Richtung
2 - Expansion in y-Richtung
3 - Expansion in x- und Y-Richtung
Beispiel: siehe SPRITE
MULTI Farbe1,Farbe2
...»Farbe1« und »Farbe2« legen die Farben zur Darstellung von Multicolor-Sprites fest. Beide Parameter müssen im Bereich 0 bis 255 liegen (s. »BORDER«).
Beispiel: siehe SPRITE
SPRITE Nummer,x,y,Block,Farbe,Modus
...läßt Sprite »Nummer« an den Koordinaten »x,y« erscheinen. Seine Daten stehen in Block und es erscheint in Farbe. Ist »Modus« = 1, so wird das Sprite in Multicolor dargestellt, wobei die durch MULTI definierten Farben verwendet werden. Wenn Modus = 0, erscheint das Sprite einfarbig. Andere Werte sind für »Modus« nicht zulässig. Die Koordinaten müssen für »x« im Bereich 0 bis 511 und für »y« zwischen 0 und 255 liegen.

```
10 MULTI 2,6
20 FOR X=0 TO 300
30 SPRITE 0,X,120,11,0,1
40 IF X=150 THEN MOBEX 0,3
50 NEXT
60 KILL 0
```

Sprite »0« bewegt sich etwa in der Mitte des Bildschirms von links nach rechts. Es erscheint im Multicolormodus in den Farben Schwarz, Rot und Blau und bezieht sein Bit-Muster aus Block 11. Nach der Hälfte des Weges wird es in beide Richtungen expandiert. Zum Schluß wird es wieder vom Bildschirm gelöscht.

Sprite-Grafik und Zeichensatzmanipulationen

Die folgenden Befehle sind sowohl auf Zeichensätze, als auch auf die Sprite-Grafik anwendbar. Der erste Parameter entscheidet über die Verwendung. Ist sein Wert »0«, bezieht sich der Befehl auf den Zeichensatz, wird er auf »1« gesetzt, ist Sprite-Grafik betroffen. Andere Werte sind nicht zulässig. »Block« bezieht sich immer auf den OFFSET 0 (s.o.), auch dann, wenn der VIC-Adreßbereich ab OFFSET 49152 beginnt. Das bedeutet:

Werden Sprites und hochauflösende Grafik und/oder RAM-Zeichensätze gleichzeitig benutzt, sind die folgenden Befehle unwirksam. Falls Sie die Befehle trotzdem benutzen, müssen die Blöcke - nach Ausführung - in die korrespondierenden Blöcke mit neuem OFFSET kopiert werden. Folgende Befehlsfolge leistet dies (s.a. DOKE):

```
DOKE 95,11*Block            (alte Anfangsadresse)
DOKE 90,11*(Block+1)        (alte Endadresse+1)
DOKE 88,49152+11*(Block+1)(neue Endadresse+1)
SYS 41919                   (Blockverschieberoutine)
```

CHANGE 0,Code,ByteNr,Byte
oder
CHANGE 1,Block,x,y,Byte

...legt im Zeichen mit Bildschirmcode »Code« (0 bis 255) das »Byte« (0 bis 255) an der durch »ByteNr« (0 bis 7) definierten Stelle ab. Für CHANGE 1 gilt: Im Sprite-Datenblock »Block« (0 bis 255) wird Byte (0 bis 255) in die durch x (0 bis 2) und y (0 bis 22) gekennzeichnete Speicherstelle geschrieben.

```
CHANGE 1,20,1,10,37
```

...legt den Wert 37 im Sprite-Datenblock 20 an Position 1,10 ab.

MOVE 0,Code1,Code2
MOVE 1,Block1,Block2

...ersetzt im Zeichensatz das Zeichen mit Bildschirmcode Code 2 (0 bis 255) durch das Zeichen mit Code 1 (0 bis 255) oder überschreibt den Sprite-Datenblock Block 2 (0 bis 255) mit den Daten aus Sprite-Datenblock Block 1 (0 bis 255).

```
MOVE 0,1,2
```

...kopiert Zeichen 1 in Zeichen 2.

NEGATE 0,Code
NEGATE 1,Block

...invertiert das Zeichen mit Bildschirmcode »Code« (0 bis 255) oder den Sprite-Datenblock »Block« (0 bis 255).

```
NEGATE 1,16
```

...invertiert den Sprite-Datenblock 16.

OVER 0,Code1,Code2
OVER 1,Block1,Block2

...verknüpft die Zeichen mit den Bildschirmcodes »Code1« und »Code2« (0 bis 255) bzw. die Daten der Sprite-Datenblöcke »Block1« und »Block2« (0 bis 255) bitweise OR. Das Ergebnis steht im Zeichen »Code2« bzw. im Spritedatenblock »Block2«.

```
OVER 1,11,23
```

...legt in Block 23 das Ergebnis der bitweisen OR-Verknüpfung von Block 11 und Block 23 ab.

Sonstige Befehle

DOKE Adresse,Wort

...legt »Wort« (0..65535) ab »Adresse« (0..65535) im Speicher ab. Dieser Befehl entspricht der Befehlsfolge POKE Adresse,Wort-256*
INT(Wort/256): POKE Adresse+1,Wort/256.

```
DOKE 55,8192
```

...setzt das Basic-RAM-Ende auf 8192.

Befehlsmodifikationen

GOSUB Rechnung oder **GOTO Rechnung**

...erlaubt bei GOTO und GOSUB eine berechnete Zeilennummer. »Rechnung« darf die erlaubten Zeilennummern nicht über, oder unterschreiten.

```
10 INPUT A
20 IF A<1 OR A>10 THEN 10
30 GOSUB 100+100*A
```

Beachten Sie daß diese Sprungziele von RENUMBER nicht korrekt umgewandelt werden!

RESTORE Zeilennummer

...setzt die DATA-Zeiger auf »Zeilennummer«.

```
RESTORE 110
```

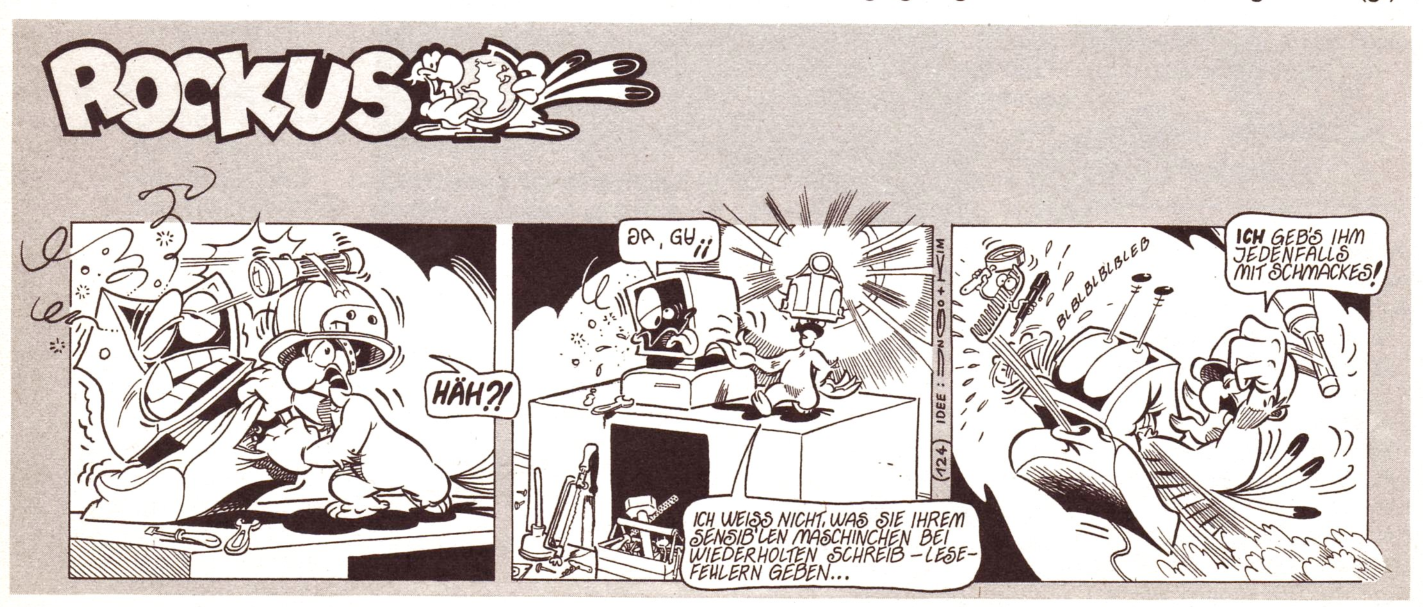
PRINT AT(x,y)

...erlaubt die formatierte Ausgabe von Texten an die durch »x« (0 bis 39) und »y« (0 bis 24) definierte Position.

```
PRINT AT(10,10) "HALLO"
```

...gibt das Wort HALLO ab Spalte 10, Zeile 10 aus.

XBasic Plus bietet Ihnen über 40 Befehle. Nach geringer Einarbeitung gelingen Ihnen die tollsten Programme. (gr)



SONDERHEFT 70

VORSCHAU

Eine wahre Fundgrube nützlicher Anwendungen

gibt es im neuen 128'er Sonderheft 70

■ »Vereinsverwaltung«, zum Erfassen und Pflegen des Mitgliederbestands jedes Vereins, Festsetzung und Überwachung der Beiträge sowie Ausdruck der Lastschriftbelege.

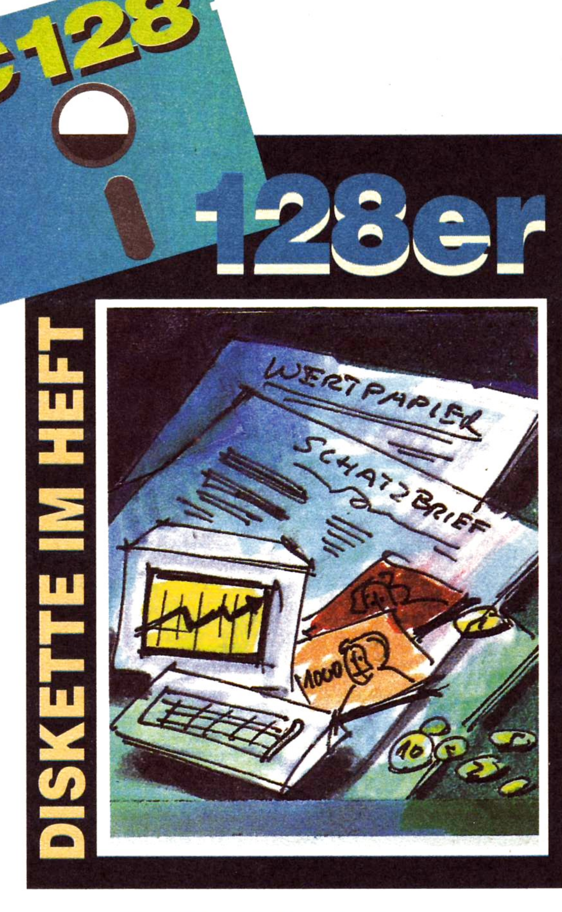
■ Wer jetzt schon wissen will, wieviel Rendite seine Geldanlage bringt, dem zeigt »Kapital« den Weg zum Reichtum. Weg mit dem Sparstrumpf.

■ Energie sparen heißt das Gebot der Stunde. Überwachen Sie mit »Energie 128« Ihren Gas-, Strom- und Wasserverbrauch.

■ Mit CP/M machen Sie mehr aus Ihrem C 128. Wie, darüber informiert Sie unser zweiter Grundlagenartikel.

■ Nützliche Tips und Tricks in Hülle und Fülle, nicht nur zur 80-Zeichen-Grafik, erwarten den C-128-Fan in diesem Heft.

■ Auch unsere Hardware-Freaks kommen nicht zu kurz: Das EPROM »RAM-Flop.Small« bietet zusätzlich 110 freie Blöcke zum Speichern beliebiger Programme und Dateien.



Nr. 70 gibt's ab 27.9.91 bei Ihrem Zeitschriftenhändler.

Aus aktuellen oder technischen Gründen können sich Themen verschieben.

---

henfolge auf und bringt die jeweiligen Variableninhalte nach Tastendruck auf den Bildschirm. Dazwischen wird erneut die Datenzeile des Hauptprogramms ausgegeben – als Beweis, daß die Routine funktioniert.

(Stephan Pätzold/bl)

Dezimaler Disassembler

Jeder Maschinensprachemonitor kennt diese Funktion: das Disassemblieren von Speicherbereichen. Die Inhalte der betreffenden Adressen werden auf dem Bildschirm gezeigt, aufgereiht wie auf einer Perlenschnur. Wer sich mit Hexzahlen nicht auskennt, sollte das Utility **minidisass** verwenden. Aus Platzgründen auf dem Bildschirm werden immer nur drei Byte-Inhalte gezeigt, rechts daneben erkennt man die ASCII-Werte. Wenn Sie das Programm gestartet haben, müssen Sie die Startadresse des gewünschten Bereichs angeben, den Sie auslesen möchten. Versuchen Sie's einmal mit Adresse 41117 (dort liegen die Texte der Basic-Befehle im Interpreter-ROM). Sie können die über den Bildschirm huschende Liste mit jeder beliebigen Taste stoppen. Das Utility eignet sich ideal zum Auslesen oder Überprüfen von Sprite-Bereichen: 3 Byte bilden immer eine Sprite-Zeile!

(Volker Hilt/bl)

Null Problemo

Treten bei Zahleneingaben oder Berechnungen Werte auf, die kleiner als »0« sind, z.B. »0,57« oder »0,89«, bringt Sie der Computer ziemlich verunstaltet: .57 oder .89 – die Null ist verschwunden! Noch schlimmer wird's, wenn die Zahl auch mit Null endet (z.B. 0,30). Die Bildschirmausgabe sieht katastrophal aus: .3!

Mit der Basic-Routine **nullen** läßt sich dieses Manko aber ausgleichen. Hat die Zahl weder vorne noch hinten eine Null, fügt das Programm beide an. Man muß berücksichtigen, daß die Zahl (gespeichert in der numerischen Variablen A) in A$ umgewandelt wird.

Ein weiterer Schwachpunkt des C 64: Er gibt Zahlen nicht bündig untereinander aus.

```
PRINT 1/10: PRINT 10
```

Sie erhalten folgende Bildschirmausgabe:

```
.1
 10
```

Mit dem Trick einer TAB-Anweisung läßt sich dieses Manko ausgleichen:

```
PRINT TAB((Z>99.99)+(Z>9
.99)+X);Z
```

X ist die Variable, die den Abstand vom linken Bildschirmrand fixiert.

Der folgende Basic-Einzeiler erzeugt in einer FOR-NEXT-Schleife jede Menge Zahlen mit Nachkommastellen:

```
10 for z=1 to 120 step .91:
   printtab((z>99.99)+(z>
9.99)
   +x);z: next
```

In Verbindung mit der Routine für die Null-Anhängsel läßt sich so eine brauchbare Bildschirmausgabe für Zahlentabellen erzeugen.

(Ingbert Nies/W. Reh/bl)

MACHT DEN



64´er - die einzige Zeitschrift, die Alles um und über den C64/C128 bringt - kompetent, umfassend, leicht verständlich und sofort anzuwenden!

Testet jetzt 64´er -

im Test-Angebot mit 3 Ausgaben für nur 19,50 DM oder holt es Euch direkt im Zeitschriftenhandel!

Diese Themen erwarten Euch in der Ausgabe 9/91:



Die besten Drucker: Großer Vergleichstest 24-Nadler bis 1500 DM und Marktübersicht



Großer Joystickteil: 6 Top-Joysticks im Test Bebilderte Marktübersicht Dynamics Interview



BTX-Modul-Erweiterung: Die Online-Kosten immer im Griff



»Dir-Printer« bringt den Inhalt Ihrer Disketten klein und mehrspaltig zu Papier



Raubkopierer-Story: Wie man sich vor Prozessen schützt



Listing des Monats: »Bundesliga V3.0« mit den Mannschaftspaarungen der Saison 91/92.

August 1991 DM 7,-

64'er 8/91

64'er TEST - ANGEBOT 3 Ausgaben für nur 19,50 DM

Ja, ich möchte 64´er Magazin mit 3 Ausgaben für 19,50 DM testen. Will ich 64´er danach weiterlesen, brauche ich nichts zu tun, ich erhalte es dann für ein Jahr zum Jahresabonnementpreis von 78,- DM, anderenfalls teile ich Ihnen dies nach Erhalt der dritten Testausgabe mit. Das Abonnement verlängert sich automatisch um ein Jahr, ich kann jederzeit zum Ende des bezahlten Zeitraumes kündigen.

Name, Vorname

Straße, Hausnummer

Postleitzahl, Wohnort

Ich bezahle
☐ nach Erhalt der Rechnung ☐ per Bankeinzug

Geldinstitut

Bankleitzahl

Kontonummer

Datum, 1. Unterschrift

Diese Vereinbarung kann ich innerhalb von acht Tagen bei der Markt&Technik Verlag AG, Postfach 1304, 8013 Haar widerrufen. Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufs. Ich bestätige die Kenntnisnahme des Widerrufsrechts durch meine 2. Unterschrift

Datum, 2. Unterschrift

03/AC 26 18

Einfach den Coupon ausfüllen und abschicken an die Markt&Technik Verlag AG, Abonnement-Service, Postfach 1304, 8013 Haar bei München.

Diese Vereinbarung können Sie innerhalb von acht Tagen bei der Markt&Technik Verlag AG, Postfach 1304, 8013 Haar widerrufen. Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufs.